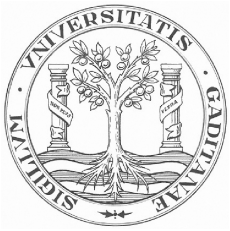


UNIVERSIDAD DE CÁDIZ
FACULTAD DE CIENCIAS

GRADO EN MATEMÁTICAS

RESOLUCIÓN NUMÉRICA DE EDP MEDIANTE ELEMENTOS FINITOS DE ORDEN ALTO Y PROGRAMACIÓN EN PARALELO

Trabajo de fin de grado presentado por

Estefanía Alonso Alonso

Tutor: Dr. J. Rafael Rodríguez Galván

Firma de la alumna

Firma del tutor

Puerto Real, Cádiz, Julio de 2016

Abstract

This project is focused on the analysis of the finite element method beginning with classical elements (defined by Lagrange basis) and continuing with finite elements with hierarchical basis (defined by Lobatto or Legendre polynomials). Taking advantage of this type of elements, we construct higher order approximations to obtain better approaches than usual ones.

On the other hand, the study of the algorithms to implement this method is applied to the development of an hybrid library in Fortran/Python. This library is used to program numerical tests whose results are in accordance with the theoretical analysis. It is also used to learn parallelization techniques oriented to the optimization of the computational cost.

A todas las personas que han hecho posible este trabajo.

Resumen

Este trabajo se centra en un análisis del método de elementos finitos que parte del análisis de elementos clásicos (formados por bases de Lagrange) y se extiende a los elementos finitos con bases jerárquicas (formadas por polinomios de Lobatto o de Legendre). Aprovechando las ventajas de éste último tipo de elementos, construimos aproximaciones de orden alto, obteniendo aproximaciones mucho mejores a las usuales.

Por otro lado, el estudio de los algoritmos para la implementación de este método se aplica al desarrollo de una biblioteca híbrida Fortran/Python. Ésta se utiliza para programar experimentos numéricos con resultados que concuerdan con el análisis teórico previo. También se usa para el aprendizaje de técnicas de paralelización orientadas a optimizar el coste computacional.

Agradecimientos

Me gustaría dar las gracias a todas las personas que me han ayudado a conseguir una de las metas más importantes de mi vida.

En primer lugar, quiero agradecer a mi familia su apoyo durante estos años, pero en especial dar las gracias a mis padres, Paco y Mª Carmen, por haber sabido guiarme por el camino correcto y hacer de mi lo que hoy soy.

Por otro lado, agradecer a mi novio, Javi, su apoyo durante estos cuatro años, estando presente tanto en los buenos como en los malos momentos apoyándome y animándome a seguir adelante.

Dar las gracias también a mis compañeros que de una forma u otra han formado parte de esta etapa de mi vida, en especial a Sandra porque más que compañeras hemos sido amigas y hemos compartido muchas experiencias que siempre estarán presentes.

Por último, aunque no menos importante, dar las gracias a mi tutor Rafa, por todo lo que me ha ayudado a sacar este trabajo adelante, enseñándome con gran dedicación esta parte tan bonita de las matemáticas.

Nadie dijo que llegar aquí fuera un camino fácil, pero vuestro ánimo y apoyo forman parte de mi éxito. Muchas gracias.

Estefanía Alonso Alonso

julio 2016

Índice general

Introducción **1**

1 Formulación variacional de EDP **5**

1.1 Espacios L^p 5

1.2 Espacios de Hilbert 6

1.2.1 Conceptos básicos 6

1.2.2 El teorema de Lax-Milgram 7

1.3 Espacios de Sobolev y formulación variacional de problemas de valor frontera 1D 8

1.3.1 Motivación. 8

1.3.2 El espacio de Sobolev $\mathbf{W^{1,p}(I)}$ 9

1.3.3 El espacio $\mathbf{W_0^{1,p}}$ 10

1.3.4 Existencia y unicidad de solución débil 11

1.3.5 Regularidad de la solución débil 12

1.3.6 Recuperar la solución clásica 13

1.4 El método de Galerkin 13

2 Elementos finitos nodales **17**

2.1 Definiciones principales 17

2.2 Mallas de elementos finitos 21

2.3 Interpolador y conformidad de elementos finitos 22

2.4 Espacios de elementos finitos continuos y discontinuos 24

2.5 Implementación del método de los elementos finitos 26

2.5.1 Aproximación a formas débiles y discretización 27

2.6 Dominios y aplicaciones de referencia 28

3 Polinomios ortogonales y elementos finitos jerárquicos **31**

3.1 Elementos finitos jerárquicos 31

3.2 Polinomios ortogonales 32

3.2.1 Polinomios de Legendre 33

3.2.2 Polinomios de Lobatto 34

4 Implementación de elementos finitos con bases jerárquicas **37**

4.1 El problema continuo y discreto 37

4.1.1 Discretización de (4.9) 38

4.1.2 Elección de la función de levantamiento $\mathbf{u}^*$ 40

4.2 Transformación al dominio de referencia $\widehat{\mathbf{K}}$ 40

4.3 Funciones de forma de alto orden 41

4.3.1 Funciones de forma de alto orden nodales 41

4.3.2 Funciones de forma de alto orden jerárquicas 42

4.3.3 El número de condición en bases jerárquicas 43

4.4 Diseño de funciones base 45

4.4.1 Funciones sombrero 46

4.4.2 Funciones burbuja 46

4.5 Estimaciones de error 46

5 Experimentos numéricos **51**

5.1 La biblioteca *libHOFFE* 51

5.2 Problema de Poisson 53

5.3 Ecuación del calor 56

5.4 Ecuación de transporte 58

5.5 Ecuación de convección-difusión 61

5.6 Problema de Poisson con bases de Lobatto de orden alto 63

5.7 Cálculo paralelo con bases de Lobatto de orden alto 65

Conclusiones y proyectos futuros **67**

A Apéndice 1 **71**

A.1 Código Python: Problema de Poisson 71

A.2 Código Python: Ecuación del calor 75

A.3 Código Python: Ecuación de transporte 76

A.4 Código Python: Ecuación de convección-difusión 79
A.5 Código Python: Problema de Poisson con bases de Lobatto de orden alto . . 81
A.6 Código Fortran: Cálculo paralelo con bases de Lobatto de orden alto 85

Bibliografía **89**

Introducción

El propósito de este trabajo es realizar un análisis detallado de la teoría del método de elementos finitos y de su implementación, abarcando tanto los elementos clásicos (con bases de Lagrange) como elementos menos comunes como son los elementos finitos con base jerárquica (formadas por polinomios de Lobatto o de Legendre).

Aprovecharemos las ventajas de este último tipo de elementos para construir aproximaciones de orden alto e investigaremos la posibilidad de usar técnicas de paralelización para reducir el coste computacional.

El interés por la resolución de ecuaciones en derivadas parciales nace del hecho de que muchos fenómenos físicos, y en general, de las ciencias experimentales como biología, economía..., pueden ser representados mediante este tipo de ecuaciones y, por ello, al resolver la ecuación podemos tanto estudiar como predecir lo que puede ocurrir con cierto fenómeno. Sin embargo, resolver este tipo de ecuaciones de forma exacta no es posible en general, por lo que intentamos encontrar una solución aproximada que se acerque en la medida de lo posible a la solución real que buscamos. Entre los métodos más usuales para la resolución de EDP de forma numérica destacamos el método de diferencias finitas y el método de elementos finitos. El primero de estos métodos es el más sencillo, pero tiene el inconveniente de su dificultad para ser aplicado en dominios n-dimensionales con geometría arbitraria.

El método de los elementos finitos que conocemos hoy apenas tiene 70 años y es el resultado de diversas contribuciones e ideas de varias generaciones de matemáticos e ingenieros. En sus inicios, el método se usaba para resolver ciertos problemas formulados generalmente en dominios de geometría compleja y que como ya hemos dicho, son difíciles de resolver mediante el método de diferencias finitas. Entre los precursores de este método encontramos a Richard Courant, que fue el primero en usar funciones polinómicas a trozos sobre una triangulación, y a John Hadji Argyris, que está considerado el padre

de los elementos finitos. A partir de 1955, el método recibe un especial interés por parte de ingenieros enfrentados a problemas de elasticidad y empiezan a definirse los elementos finitos clásicos. Sin embargo, el análisis matemático del método no comienza a desarrollarse hasta 10 años después de la mano de la obra de G. Strang y G. Fix, así como de la obra de P.G. Ciarlet. El lector interesado en conocer más detalles sobre este asunto puede consultar [2].

En este trabajo nos centramos en estudiar detalladamente los conceptos matemáticos que definen el método de los elementos finitos, para así poder hacer un análisis de forma rigurosa. El conocimiento adquirido nos permitirá, a su vez, extender los conceptos usualmente relacionados con el método de elementos finitos y así abarcar casos que son menos utilizados en la práctica pero que tienen un interés relevante desde el punto de vista matemático, a la vez de importantes ventajas prácticas, que serán investigadas en el presente trabajo.

Este método se basa en la idea principal de dividir el dominio en diferentes partes llamadas elementos, así como en la definición de espacios finitos dimensionales, $V_{h,p}$, formados por funciones polinómicas a trozos de grado menor o igual que p definidas en cada uno de estos elementos. En cada uno de ellos se aproxima la solución exacta escribiéndola como combinación lineal de los elementos de una base $\mathcal{B}^p$ del espacio $V_{h,p}$.

Bajo determinadas hipótesis de regularidad de los datos se demuestra que cuando h (el diámetro de los elementos) tiende a 0, la solución aproximada converge a la solución exacta. Esto mismo ocurre cuando el grado de los polinomios, p, tiene a infinito. Sin embargo, no siempre podemos disponer de la potencia de cálculo necesaria para calcular la solución aproximada con h arbitrariamente pequeño y p tan grande como deseemos, puesto que la implementación de los elementos finitos requiere de la resolución de sistemas de ecuaciones cuyas dimensiones crecen cuando h tiende a 0 o p tiende a infinito.

Normalmente, la base $\mathcal{B}^p$ está formada por polinomios de interpolación de Lagrange en nodos definidos dentro del elemento. Este tipo de elementos clásicos son conocidos como elementos finitos de Lagrange.

Pero más allá de éstos y dentro de la definición abstracta de elemento finito que veremos más adelante, existen numerosas familias de elementos, cuyo estudio excede las pretensiones de este trabajo. Por ello, nos centraremos en una familia en particular que presenta un interés destacable tanto de forma teórica como práctica. Se trata de los elementos finitos con bases jerárquicas, que serán aquellas que verifiquen $\mathcal{B}^p \subset \mathcal{B}^{p+1}$. Más todavía,

dentro de estos elementos, nos centraremos en aquellos cuyas bases o las derivadas de éstas sean ortogonales. Esto nos llevaría a la resolución de un sistema de ecuaciones con matriz diagonal, que se traduce en un ahorro considerable del esfuerzo computacional requerido por el método de los elementos finitos, dando pie al uso de técnicas de cálculo en paralelo tanto para el montaje de las matrices como para la resolución del sistema de ecuaciones.

Además de lo anterior, existen diversos motivos que justifican que centremos nuestra atención en este tipo de elementos:

Por un lado, el interés por el aprendizaje de técnicas matemáticas que extienden las competencias abarcadas por las asignaturas del Grado en Matemáticas.

Por otro lado, aunque existen numerosas bibliotecas informáticas especializadas en el tema, como por ejemplo [4], existe un gran interés en estudiar los algoritmos relacionados con la implementación a bajo nivel del método de elementos finitos (el montaje de las estructuras de datos relacionadas con los sistemas de ecuaciones). Además, salvo casos excepcionales estas bibliotecas utilizan exclusivamente elementos finitos de Lagrange, mientras que nuestro interés reside en los elementos finitos jerárquicos. Su resolución en la práctica nos va a ayudar a entender mucho mejor el método y las técnicas que se utilizan, además de introducirnos en el campo de técnicas avanzadas de computación como es el cálculo en paralelo, que posibilite la resolución de los sistemas con un menor coste computacional. Para ello, se ha desarrollado una biblioteca Fortran/Python como veremos a lo largo del trabajo.

Por tanto, puesto que en el trabajo nos proponemos tanto hacer un estudio teórico detallado como desarrollar una implementación práctica de éste, los objetivos que nos proponemos son ambiciosos. Por este motivo, se ha optado por tomar como modelo el caso de los elementos finitos en dimensión 1, pero siempre a través de un enfoque que hace énfasis en su inmediata generalización al caso n-dimensional.

Este trabajo está estructurado de la siguiente forma:

En el primer capítulo vemos una introducción a la teoría variacional de las EDP así como al Método de Galerkin. Para ello, es necesario introducir los espacios de Sobolev y el teorema de Lax-Milgram, que será de gran utilidad para el análisis de existencia y unicidad de solución.

En el segundo capítulo introducimos las nociones relacionadas con los elementos finitos, que serán necesarias para entender el resto de este trabajo. Debe señalarse que la

forma de analizar este método es más general que la habitual, pues se trata de abarcar espacios más amplios que los clásicos elementos de Lagrange y se introducen conceptos como elementos finitos discontinuos o conformidad.

En el tercer capítulo estudiamos las bases jerárquicas y los polinomios de Legendre y de Lobatto. A partir de aquí, podemos hacernos una idea de las propiedades que los elementos finitos con este tipo de bases pueden alcanzar.

En el cuarto capítulo tenemos un ejemplo en el cual podemos comparar el desarrollo del método para un problema en dimensión 1 para elementos clásicos (con bases de Lagrange) y jerárquicos (con bases de Lobatto). Este ejemplo es muy amplio de forma que haya la menor pérdida de generalidad posible.

En el quinto capítulo hemos hecho distintos experimentos numéricos utilizando una biblioteca Fortran/Python que hemos programado. En estos ejemplos abarcamos tanto ecuaciones en derivadas parciales de tipo elíptico, parabólico e hiperbólico. Aquí aplicamos todo lo estudiado teóricamente en los capítulos precedentes con el objetivo de comprobar que todo funciona y que los resultados son los previstos. Se incluyen ejemplos 1d en el que se alcanza el orden 7 en los polinomios (alcanzando la solución exacta para la precisión del ordenador).

Por último plantemos las conclusiones y proyectos futuros que se pueden extraer de todos los capítulos mencionados. Estos serán muy numerosos ya que desde el principio se ha planteado como un trabajo abierto a su posible extensión futura hacia campos muy diversos.

CAPITULO

1

Formulación variacional de EDP

Este capítulo está dedicado a una introducción a la formulación variacional de las ecuaciones en derivadas parciales y al método de Galerkin. Para ello haremos un breve estudio de los espacios de Sobolev. Los resultados que aquí se muestran son necesarios para poder definir conceptos y justificar resultados en los siguientes capítulos. Para profundizar más en el tema ver [1].

1.1 Espacios L^p

En lo que sigue, Ω denotara a un conjunto abierto de $\mathbb{R}^N$. Sea (Ω, M, μ) un espacio medible, esto es:

1. M es un $\sigma-$álgebra en Ω,
2. μ es medible,
3. Ω es $\sigma-$finito.

Denotamos por $L^1(\Omega, \mu)$, o simplemente $L^1(\Omega)$ al espacio de las funciones de Ω en $\mathbb{R}$ que son integrables. En general:

Definición 1.1. Sea $p \in \mathbb{R}$ con $1 < p < \infty$. Definimos

$$L^p(\Omega) = \{f : \Omega \longrightarrow \mathbb{R} : f \text{ es medible y } |f|^p \in L^1(\Omega)\}. \tag{1.1}$$

Para cada $f \in L^p(\Omega)$, $1 \leq p < \infty$, se define $||f||_{L^p} = ||f||_p = \left(\int_\Omega |f(x)|^p d\mu \right)^{1/p}$.

Definición 1.2. Definimos

$$L^\infty(\Omega) = \{f : \Omega \longrightarrow \mathbb{R} : f \text{ es medible y } \exists c > 0 \text{ tal que } |f(x)| \leq c \text{ c.p.t. } x \in \Omega\}, \quad (1.2)$$

y, para cada $f \in L^p(\Omega)$, $||f||_{L^\infty} = ||f||_\infty = \inf\{c \in \mathbb{R} : |f(x)| \leq c \text{ c.p.t. } x \in \Omega\}$.

Se puede demostrar que $||\cdot||_p$ es una norma para $1 \leq p \leq \infty$. La demostración se basa en la desigualdad de Hölder:

$$\int |fg| \leq ||f||_p ||g||_{p'}, \quad \forall f \in L^p, g \in L^{p'} \text{ con } \frac{1}{p} + \frac{1}{p'} = 1.$$

A continuación se exponen algunas propiedades importantes de los espacios $L^p(\Omega)$. Su demostración puede consultarse, por ejemplo, en [1].

Teorema 1.3. Si Ω de es medida finita y si $1 \leq p < q \leq \infty$, entonces $L^q(\Omega) \subset L^p(\Omega)$ y existe $C > 0$ tal que

$$||f||_p \leq C||f||_q \quad \forall f \in L^q(\Omega).$$

Teorema 1.4. $L^p(\Omega)$ es un espacio de Banach para todo $1 \leq p \leq \infty$.

Para cada $l = 1, 2, ...$, incluyendo $l = \infty$, denotamos por $C_c^l(\Omega)$ al espacio de las funciones de clase l en Ω con soporte compacto. Esto es,

$$C_c^l(\Omega) = \{f \in C^l(\Omega) : f(x) = 0 \, \forall x \in \mathbb{R}^N \backslash K \text{ donde } K \text{ es compacto}\}.$$

Con frecuencia se denota $C_c(\Omega) = C_c^0(\Omega)$ y $D(\Omega) = C_c^\infty(\Omega)$. Entonces se tiene:

Teorema 1.5. El espacio $C_c^\infty(\Omega)$ es denso en $L^p(\Omega)$, para todo $1 \leq p < \infty$; es decir,

$$\forall f \in L^p(\Omega), \, \forall \epsilon > 0 \, \exists f_1 \in C_c(\Omega) \text{ tal que } ||f - f_1||_p \leq \epsilon.$$

Teorema 1.6. Dada $u \in L^1(\Omega)$ tal que

$$\int_\Omega uf = 0 \text{ para todo } f \in C_c^\infty(\Omega),$$

entonces $u = 0$ c.p.d. en Ω.

1.2 Espacios de Hilbert

1.2.1 Conceptos básicos

Definición 1.7. Sea H un espacio vectorial. Un *producto escalar* (u, v) es una forma bilineal en $H \times H$ con valores en $\mathbb{R}$ tal que:

- $(u, v) = (v, u)\ \forall u, v \in H$ (simétrica)
- $(u, v) \geq 0\ \forall u \in H$ (positiva)
- $(u, u) \neq 0\ \forall u \neq 0$ (bien definida)

Es bien sabido que el producto escalar satisface la desigualdad de Cauchy-Schwarz:

$$|(u, v)| \leq (u, u)^{1/2}(v, v)^{1/2}, \quad \forall u, v \in H. \tag{1.3}$$

Como consecuencia, la expresión

$$||u|| = (u, u)^{1/2} \tag{1.4}$$

es una norma, llamada norma inducida por el producto escalar.

Definición 1.8. Un *espacio de Hilbert* es un espacio vectorial H dotado de un producto escalar y tal que H es completo[1] para la norma inducida por este producto escalar.

Ejemplo 1.9. El espacio $H = L^2(a, b) = \{f : [a, b] \longrightarrow \mathbb{R}$ medibles tal que $\int_a^b |f(x)|^2 dx < +\infty\}$ es de Hilbert para el siguiente producto escalar:

$$(f, g) = \int_a^b f(x)g(x)dx,$$

que está bien definido gracias a la desigualdad de Hölder. La norma inducida es

$$||f||_{L^2(a,b)} = (f, f)^{1/2} = \left(\int_a^b |f(x)|^2 dx\right)^{1/2},$$

que coincide con la norma introducida en la definición 1.1 (para $p = 2$ y $\Omega = (a, b)$).

Observación 1.10.

1. El ejemplo anterior, en general, se puede extender a conjuntos $\Omega \subset \mathbb{R}^n,\ n \geq 2$
2. El espacio H^1 que estudiaremos más adelante es también un espacio de Hilbert.

1.2.2 El teorema de Lax-Milgram

Definición 1.11. Una forma bilineal $a : H \times H \longrightarrow \mathbb{R}$ donde H es un espacio de Hilbert es:

- *continua*, si existe una constante $c > 0$ tal que

$$|a(u, v)| \leq c|u||v|, \quad \forall u, v \in H. \tag{1.5}$$

[1] Recordemos que un espacio es completo cuando todas las sucesiones de Cauchy son convergentes.

- *coerciva*, si existe una constante $\alpha > 0$ tal que

$$a(u,v) \geq \alpha |v|^2, \quad \forall v \in H. \tag{1.6}$$

Teorema 1.12. (Lax-Milgram)

Sea $a(u,v)$ una forma bilineal, continua y coerciva en un espacio de Hilbert, H y sea $L : H \longrightarrow \mathbb{R}$ lineal y continua. Entonces, existe un único $u \in H$ tal que

$$a(u,v) = L(v), \quad \forall v \in H. \tag{1.7}$$

1.3 Espacios de Sobolev y formulación variacional de problemas de valor frontera 1D

1.3.1 Motivación.

Para una mayor claridad, el desarrollo de este apartado se planteará en el marco unidimensional. De todas formas, estos resultados se podrían generalizar al campo n-dimensional (ver [1]). Consideremos como modelo el siguiente problema: dada $f \in C([a,b])$, hallar una función u que satisfaga:

$$\begin{cases} -\Delta u + u = f & \text{en } [a,b], \\ u(a) = u(b) = 0. \end{cases} \tag{1.8}$$

En el caso unidimensional, el operador laplaciano, Δu, coincide con la derivada segunda, u'' y una solución clásica de este problema es una función $u \in C^2([a,b])$ tal que $u(x)$ satisface (1.8) para todo $x \in [a,b]$. Sea $V = C^1([a,b])$ tal que $v(a) = v(b) = 0$ para todo $v \in V$. Multiplicando (1.8) por cualquier función $v \in V$ e integrando por partes tenemos:

$$\int_a^b u'v' + \int_a^b uv = \int_a^b fv \quad \forall v \in V. \tag{1.9}$$

Para que esta expresión tenga sentido es suficiente que $u \in C^1([a,b])$. Así, provisionalemente, podríamos decir que una función $u \in V$ que satisface (1.9) es una solución débil de (1.8).

Sin embargo, no podemos garantizar la existencia y unicidad de solución utilizando directamente este espacio $V \subset C^1([a,b])$. Esto es debido a que en este marco perdemos la posibilidad de utilizar el teorema de Lax-Milgram, ya que el espacio V no es un espacio de Hilbert (no es difícil probar que este espacio no es completo para la norma asociada al producto escalar de $L^2(a,b)$ o para el producto escalar "natural" $(u,v) = \int_a^b uv + \int_a^b u'v'$).

Necesitamos, por lo tanto, utilizar un espacio de funciones más amplio que $C^1([a,b])$. Para ello utilizamos el hecho de que (usando Cauchy-Schwarz) en (1.9) se podría relajar la restricción $u, v \in C^1([a,b])$ imponiendo sólo que verifiquen

$$u, v \in L^2(a,b) \text{ y } u', v' \in L^2(a,b). \tag{1.10}$$

Para precisar estos conceptos introducimos la teoría variacional de las ecuaciones en derivadas parciales que se basa en las siguientes etapas:

A. Precisar la noción de solución débil de (1.8). Para ello introduciremos los espacios de Sobolev.

B. Demostrar la existencia y unicidad de solución débil. Para ello veremos que se verifican las hipótesis del teorema de Lax-Milgram.

C. Probar que bajo unas hipótesis de regularidad mínimas la solución débil es de clase C^2.

D. Recuperar la solución clásica. Probar que cada solución débil que sea C^2 es solución clásica.

A continuación, nos disponemos a desarrollar cada uno de estos apartados:

1.3.2 El espacio de Sobolev $\mathbf{W^{1,p}(I)}$

Por brevedad, redactamos esta sección para el caso 1d, aunque todos los resultados son aplicables a un dominio abierto $\Omega \subset \mathbb{R}^N$.

Sea $I = (a,b)$ un intervalo abierto y sea $p \in \mathbb{R}$ con $1 \leq p \leq \infty$.

Definición 1.13. El espacio de Sobolev $W^{1,p}(I)$ se define como:

$$W^{1,p}(I) = \{u \in L^p(I) : \exists g \in L^p(I) \text{ tal que } \int_I u\varphi' = -\int_I g\varphi,\ \forall \varphi \in C_c^1(I)\}. \tag{1.11}$$

En este caso, denotamos $u' = g$ (bien definida c.p.d. en I, véase [1]) y decimos que u' es la *derivada débil o variacional* de u.

En particular, se define $H^1(I) = W^{1,2}(I)$.

Observación 1.14.

1. Se puede demostrar que los espacios $W^{1,p}$ con $1 \leq p \leq \infty$ equipados con la norma $||u||_{W^{1,p}} = ||u||_{L^p} + ||u'||_{L^p}$ (o bien, si $1 < p < \infty$, con la norma equivalente $||u||_{W^{1,p}} = (||u||_{L^p}^p + ||u'||_{L^p}^p)^{1/p}$ son espacios de Banach.

2. En el caso particular de H^1, con el siguiente producto escalar

$$(u, v)_{H^1} = (u, v)_{L^2} + (u', v')_{L^2} = \int_a^b (uv + u'v'),$$

y con la norma asociada

$$||u||_{H^1} = (||u||_{L^2}^2 + ||u'||_{L^2}^2)^{1/2}, \tag{1.12}$$

no es difícil demostrar que es un espacio de Hilbert.

3. Dados un entero $m \geq 2$ y un número real $1 \leq p \leq \infty$, se define por recurrencia el espacio

$$W^{m,p}(I) = \{u \in W^{m-1,p}(I) : u' \in W^{m-1,p}(I)\},$$

y se denota $H^m(I) = W^{m,2}(I)$.

4. Las funciones continuas son densas en $W^{m,p}$. Más aún, se tiene el siguiente resultado (ver [3]): dado un abierto $\Omega \subset \mathbb{R}^N$, el espacio $C^\infty(\Omega) \cap W^{m,p}(\Omega)$ es denso en $C^\infty(\Omega)$ para todo $1 \leq p < \infty$.

1.3.3 El espacio $\mathbf{W_0^{1,p}}$

Dado $1 \leq p < \infty$, denotamos por $W_0^{1,p}(I)$ a la clausura de $C_c^1(I)$ en $W^{1,p}(I)$. Denotamos además, $H_0^1(I) = W_0^{1,2}(I)$. Dotamos al espacio $W_0^{1,p}(I)$ con la norma de $W^{1,p}(I)$ y al espacio H_0^1 con el producto escalar de H^1.

Observación 1.15. Se puede comprobar que:

1. El espacio $W_0^{1,p}$ es un espacio de Banach.

2. El espacio H_0^1 es un espacio de Hilbert para la norma de H^1, o bien, para la siguiente norma

$$||u||_{H^1} = ||u'||_{L^2}. \tag{1.13}$$

Esta norma es equivalente a la norma (1.12) debido a la siguiente desigualdad, conocida como desigualdad de Poincaré (ver [1]):

Sea I un intervalo acotado entonces existe una constante c tal que

$$||u||_{H^1} \leq c||u'||_{L^2}, \quad \forall u \in H_0^1(I). \tag{1.14}$$

El siguiente resultado, cuya demostración se puede ver en [1], nos permite recuperar las condiciones de contorno de nuestro problema.

Teorema 1.16. Sea $u \in W^{1,p}(I)$. Entonces, $u \in W_0^{1,p}(I)$ si y solo si $u(a) = u(b) = 0$.

Observación 1.17. El teorema anterior se podría generalizar a dimensión n y significa que las funciones de $W_0^{1,p}(\Omega)$ son aquellas funciones de $W^{1,p}(\Omega)$ que son 0 en $\partial\Omega$. Hay que tener en cuenta que no es trivial definir la restricción de una función de $W^{1,p}(\Omega)$ a la $\partial\Omega$, que es un conjunto de medida nula. A esta restricción se le llama *traza* de la función. Los detalles se escapan de los objetivos de este trabajo, el lector interesado puede consultar [1], capítulo 9.

Una vez estudiados los espacios de Sobolev, podemos precisar el concepto de solución débil, que era el objeto de este apartado. Para ello definimos:

Definición 1.18. Escribimos la *formulación variacional* del problema (1.8) como: Hallar $u \in H_0^1([a,b])$ que verifica

$$\int_a^b u'v' + \int_a^b uv = \int_a^b fv, \ \ \forall v \in H_0^1([a,b]). \tag{1.15}$$

En este caso, decimos que u es una *solución débil* del problema (1.8).

1.3.4 Existencia y unicidad de solución débil

En este momento, contamos con las herramientas matemáticas necesarias para estudiar la existencia y unicidad de solución débil para el problema (1.8). Para ello, denotamos

$$a(u,v) = \int_a^b u'v' + \int_a^b uv,$$
$$L(v) = \int_a^b fv.$$

Comprobaremos que se verifican las hipótesis del teorema de Lax-Milgram.

1. Veamos que $a(\cdot,\cdot)$ es continua:

$$a(u,v) = (u,v)_{H^1} \leq (u,u)_{H^1}^{1/2}(v,v)_{H^1}^{1/2} = ||u||_{H^1}||v||_{H^1}. \tag{1.16}$$

2. Veamos que $a(\cdot,\cdot)$ es coerciva:

$$a(v,v) = \int_a^b v^2 + \int_a^b (v')^2 = ||v||_{H^1}^2. \tag{1.17}$$

3. Veamos que L es continua:

$$L(v) = \int_I fv = (f,v)_{L^2} \leq (f,f)^{1/2}(v,v)^{1/2} = ||f||_{L^2}||v||_{L^2} \leq ||f||_{L^2}||v||_{H^1} = C||v||_{H^1}. \tag{1.18}$$

1.3.5 Regularidad de la solución débil

Una vez estudiada la existencia y unicidad de solución débil, nos disponemos a estudiar la regularidad de ésta. En concreto: si los datos del problema son suficientemente regulares entonces la solución débil es de clase $C^2(\Omega)$.

La demostración rigurosa, en el caso n–dimensional, de la afirmación anterior no es sencilla y requiere de una serie de conceptos relacionados con la teoría de espacios de Sobolev cuyo estudio se escapa de las pretensiones de este trabajo. El lector interesado puede consultar la sección 6 del capítulo 9 de [1]. En cualquier caso, a continuación desarrollaremos la idea de la demostración en el caso unidimensional.

Sea u la solución débil de (1.8), es decir, $u \in H_0^1([a,b])$ que verifica (1.15). Entonces, integrando por partes[1] tenemos que

$$-\int_a^b u''v + \int_a^b uv = \int_a^b fv, \quad \forall v \in H_0^1([a,b]). \tag{1.19}$$

Es decir, u'' es una función que verifica

$$\int_a^b (-u'' + u - f)v = 0, \quad \forall v \in H_0^1(\Omega). \tag{1.20}$$

En particular, como $C_c^\infty(\Omega) \subset H_0^1(\Omega)$,

$$\int_a^b (-u'' + u - f)v = 0, \quad \forall v \in C_c^\infty(\Omega) \tag{1.21}$$

y, debido[2] al teorema 1.6:

$$-u'' + u - f = 0 \ \text{ c.p.d. en } \Omega. \tag{1.22}$$

De aquí tenemos que

$$u'' = u - f \in L^2(\Omega) \ \text{ c.p.d. en } \Omega, \tag{1.23}$$

es decir, u'' se puede identificar con una función de $L^2(\Omega)$ y así $u \in H^2(\Omega)$.

[1] Obsérvese que, puesto que aún no tenemos que $u'' \in L^2(a,b)$, esta fórmula de integración no tiene sentido. Para poder dotar de rigor a esta expresión, es necesario definir una fórmula de integración en un espacio más amplio que $H^2(\Omega)$, en el que tenga sentido la expresión $\int_a^b u''v$. Este tipo de espacios, conocidos como espacios de distribuciones, se definen a partir de las aplicaciones lineales y continuas sobre $C_c^\infty(\Omega)$. El lector interesado puede consultar por ejemplo [1].

[2] En realidad, a una generalización de este teorema que tenga sentido en un marco más amplio, el marco de las distribuciones, véase [1]

Como habíamos pedido ciertas hipótesis de regularidad sobre los datos del problema podemos, en concreto, tomar $f \in C^0([a,b])$. Además en dimensión 1, se verifica que $H^1(\Omega) \subset C^0(\Omega)$. Esto es consecuencia de unos resultados clásicos en la teoría de espacios de Sobolev, conocidos como inyecciones de Sobolev (ver [1]). Por tanto, $u \in C^0([a,b])$ y en consecuencia, $u'' \in C^0([a,b])$. De aquí tenemos que $u \in C^2([a,b])$.

1.3.6 **Recuperar la solución clásica**

Por lo visto en la sección anterior podemos admitir que la solución débil $u \in H_0^1(\Omega)$ del problema (1.8) es de clase $C^2(\Omega)$ para datos suficientemente regulares (para $f \in C(\Omega)$, aunque en dimensión dos o mayor también hay que suponer cierta regularidad en la frontera de Ω). Como se verifica que:

$$\int_\Omega (-\Delta u + u - f)v = 0, \ \ \forall v \in C_c^1(\Omega), \tag{1.24}$$

entonces

$$-\Delta u(x) + u(x) - f(x) = 0, \ \ \forall x \in \Omega. \tag{1.25}$$

Comprobaremos este resultado directamente, aunque también se podría utilizar el teorema 1.6.

Demostración. Supongamos que existe $x_0 \in \Omega$ tal que $-\Delta u(x_0) + u(x_0) - f(x_0) > 0$ (se haría de forma análoga suponiendo $-\Delta u(x_0) + u(x_0) - f(x_0) < 0$), entonces existe un $\delta > 0$ tal que $-\Delta u(x) + u(x) - f(x) > 0$ para todo $x \in (x_0 - \delta, x_0 + \delta)$.

Sea ahora $v \in C_c^1(\Omega)$ tal que $v > 0$ en $(x_0 - \delta, x_0 + \delta)$ y $v = 0$ en $\Omega \setminus (x_0 - \delta, x_0 + \delta)$. Entonces

$$\int_\Omega (-\Delta u + u - f)v > 0, \tag{1.26}$$

lo cual es una contradicción por (1.24). □

1.4 El método de Galerkin

Consideremos el siguiente problema abstracto:

$$\begin{cases} \text{Hallar } u \in V \text{ tal que,} \\ a(u,v) = f(v), \qquad \forall v \in V. \end{cases} \tag{1.27}$$

Donde a es una forma bilineal continua, f es una forma lineal continua y V es un espacio de Hilbert. Supongamos, por tanto, que este problema está bien planteado, por ejemplo, porque a es coerciva.

La idea principal de método de Galerkin es aproximar V por un espacio de dimensión finita V_h, de forma que la norma que definimos sobre este último sea la misma que la que se define para V.

Aunque se puede ver de forma más general (ver [5]), definiremos el método de Galerkin de la siguiente forma: El método de Galerkin trata de construir una aproximación de u (solución de 1.27) mediante la resolución del siguiente problema:

$$\begin{cases} \text{Hallar } u_h \in V_h \text{ tal que,} \\ a_h(u_h, v_h) = f_h(v_h), \qquad \forall v_h \in V_h. \end{cases} \tag{1.28}$$

Antes de continuar enunciaremos algunas definiciones que posteriormente nos harán falta conocer.

Definición 1.19. El conjunto V_h se denomina *conforme* si $V_h \subset V$.

Mientras que no se especifique lo contrario, consideraremos aplicaciones conformes.

Definición 1.20. El conjunto V_h se denomina *consistente* si la solución exacta u verifica el problema aproximado, esto es,

$$a_h(u, v_h) = f_h(v_h), \quad \forall v_h \in V_h. \tag{1.29}$$

Para proseguir con el método de Galerkin queremos ver que el problema aproximado es equivalente a un sistema lineal. Para ver esto, denotamos $N = \dim V_h$ y sea $\{\phi_1, ..., \phi_N\}$ una base de V_h. En el contexto de los elementos finitos, las funciones de la base pueden ser tomadas como funciones globales de forma en V_h (ver Capítulo 2). Escribimos u_h como combinación de elementos de la base

$$u_h = \sum_{i=1}^{N} U_i \phi_i, \tag{1.30}$$

donde $U = (U_i)_{1 \leq i \leq N} \in \mathbb{R}^N$ es el vector de coordenadas de u_h. Sea $\mathcal{A} \in \mathbb{R}^{N \times N}$ definida por

$$\mathcal{A}_{ij} = a_h(\phi_i, \phi_j), \;\; 1 \leq i \leq N, \; 1 \leq i \leq N, \tag{1.31}$$

a la que llamaremos *matriz de rigidez (generalizada)* y sea $\mathcal{F} \in \mathbb{R}^N$ con

$$\mathcal{F}_i = f_h(\phi_i), \;\; 1 \leq i \leq N. \tag{1.32}$$

Se puede comprobar fácilmente que u_h verifica (1.28) si y sólo si U es solución del sistema de ecuaciones lineales

$$\mathcal{A}U = \mathcal{F}. \tag{1.33}$$

Véase que si $a_h(\cdot,\cdot)$ es simétrica, la matriz del sistema anterior también es simétrica. Además, se puede relacionar que la coercividad de $a(\cdot,\cdot)$ con el hecho de que A sea definida positiva y por tanto tenga buenas propiedades de cara a los métodos numéricos para la resolución del problema anterior. En todo caso, en este momento es muy fácil deducir el siguiente resultado:

Proposición 1.21. Sea V un espacio de Hilbert, sea a una forma bilineal continua y sea f una forma lineal continua. Si V_h es un espacio finito dimensional y se tiene que

- a es coerciva en V,
- $V_h \subset V$.

Entonces el problema aproximado (1.28) está bien definido. En particular, existe una única solución del sistema de ecuaciones (1.33).

Demostración. Este resultado es claro pues estamos bajo las condiciones del teorema de Lax-Milgram (1.7) y, por construcción, la existencia de solución de (1.33) es equivalente a la existencia de solución del sistema lineal (1.28). □

CAPITULO

2

Elementos finitos nodales

Este capítulo establecerá las bases para el estudio del método de los elementos finitos. Aquí se plantean las definiciones y los resultados matemáticos que son de vital importancia para su posterior aplicación. Además, de define la familia de elementos "clásica" conocida como elementos (nodales) de Lagrange.

2.1 Definiciones principales

Definición 2.1. Un *elemento finito* en el sentido de Ciarlet [6] es una terna $\mathcal{K} = (K, P, \Sigma)$ donde:

- K es un dominio cerrado, con interior no vacío en $\mathbb{R}^d$. En concreto, dependiendo de la dimensión, consideraremos intervalos (d=1), triángulos o cuadriláteros (d=2), etc...
- P es un espacio de polinomios en K de dimensión N_P.
- $\Sigma = \{L_1, L_2, ..., L_{N_P}\}$ es un conjunto de formas lineales y continuas, $L_i : P \to \mathbb{R}$ para $i = 1, .., N_P$, tales que la aplicación lineal

$$\begin{aligned} L : P &\longrightarrow \mathbb{R}^{N_P}, \\ q &\longmapsto L(q) = (L_1(q), L_2(q), ..., L_{N_P}(q)) \end{aligned}$$

es biyectiva. A los elementos de Σ se les llama *grados de libertad*.

Observación 2.2. Esta última condición significa que para todo $(\alpha_1, ..., \alpha_{N_P}) \in \mathbb{R}^{N_P}$ existe un único $q \in P$ tal que $L_i(q) = \alpha_i$ para $1 \leq i \leq N_P$. Pero ya que hemos supuesto que $dimP = card\Sigma = N_P$, lo anterior es equivalente a la siguiente condición:

> Para todo $q \in P$ se verifica que si $L_1(q) = L_2(q) = ... = L_{N_P}(q) = 0$, entonces $q = 0$ (es decir, la aplicación lineal L es inyectiva).

Esta propiedad (biyectividad de L) se denomina *unisolvencia*. Con frecuencia, para enfatizar esta propiedad, diremos que (K, P, Σ) es un elemento finito unisolvente (en concordancia con parte de la literatura especializada en el tema, como [7]). En cualquier caso, en nuestro trabajo, si un elemento no verifica la unisolvencia entonces no es un elemento finito.

Proposición 2.3. Sea $\mathcal{K} = (K, P, \Sigma)$ un elemento finito con $dim(P) = N_P$. Entonces, existe una base $\{\theta_1, ..., \theta_{N_P}\}$ en P tal que:

$$L_i(\theta_j) = \delta_{ij}, \quad 1 \leq i, j \leq N_P. \tag{2.1}$$

En este caso, decimos que la base $\{\theta_1, ..., \theta_{N_P}\}$ verifica la δ-*propiedad*.

Demostración. Esta demostración es consecuencia directa de la biyectividad del operador L. En concreto: Para cada $j = 1, ..., N_P$ tomamos el vector v_j como el vector que tiene un 1 en la posición j y 0 en las demás. Entonces, existe un único $\theta_j \in P$ tal que $L(\theta_j) = v_j$, es decir,

$$(L_1(\theta_j), ..., L_i(\theta_j), ..., L_N(\theta_j)) = (0, ..., \underset{j}{1}, ..., 0),$$

esto es, $L_i(\theta_j) = \delta_{ij}$.

Además $\{\theta_j\}_1^{N_P}$ es base de P. Como ambos conjuntos tienen la misma dimensión, es suficiente que los vectores sean linealmente independientes, en efecto,

$$\sum_{j=1}^{N_P} \alpha_j \theta_j = 0 \Rightarrow L_i(\sum_{j=1}^{N_P} \alpha_j \theta_j) = 0 \Rightarrow \sum_{j=1}^{N_P} \alpha_j L_i(\theta_j) = 0 \Rightarrow \alpha_i = 0, \ \ \forall i = 1, ..., N_P.$$

□

Definición 2.4. Las $\{\theta_1, .., \theta_{N_P}\}$ anteriores se denominan *funciones de forma locales*.

Definición 2.5. Sea $K = [a, b] \subset \mathbb{R}$ y sea $P = \mathcal{P}_p(K)$ el espacio de polinomios de orden[1] p en K. Definimos $N_P = p + 1$ y elegimos N_P puntos en K, $a = x_1 \leq x_2 \leq ... \leq x_{N_P} = b$.

[1] En este trabajo llamaremos polinomios de orden p a los polinomios de orden menor o igual que p.

Sean $\{\mathcal{L}_1^p, ..., \mathcal{L}_{N_P}^p\}$ los *polinomios de Lagrange* asociados a la partición anterior definidos como

$$\mathcal{L}_i^p(x) = \frac{\prod\limits_{i\neq j}(x - x_j)}{\prod\limits_{i\neq j}(x_i - x_j)}, \quad 1 \leq i \leq N_P. \tag{2.2}$$

Entonces decimos que un elemento finito (K, P, Σ) es de *Lagrange*, si verifica que $\{\mathcal{L}_1^p, ..., \mathcal{L}_{N_P}^p\}$ es la base de P y además el conjunto de grados de libertad $\Sigma = \{L_1, L_2, ..., L_{N_P}\}$ verifica

$$\begin{aligned} L_i : P &\longrightarrow \mathbb{R}, \\ q &\longmapsto q(x_i), \end{aligned}$$

para todo $q \in P$ e $i = 1, ..., N_P$.

A este tipo de elementos se les conoce como *nodales* debido a que el valor del i–ésimo grado de libertad $L_i(q)$ coincide con $q(x_i)$, el valor de q en el nodo x_i.

Ejemplo 2.6. (Elemento finito unisolvente)
Consideramos un elemento finito de Lagrange con $p = 1$ sobre el intervalo $K = [-1, 1]$. Tenemos entonces que $P = \mathcal{P}_1(K)$ y la partición se reduce a $x_1 = -1$ y $x_2 = 1$, los extremos del intervalo. Sean $\alpha, \beta \in \mathbb{R},\ q_1, q_2 \in P$ entonces

$$L_i(\alpha q_1 + \beta q_2) = (\alpha q_1 + \beta q_2)(x_i) = \alpha q_1(x_i) + \beta q_2(x_i) = \alpha L_i(q_1) + \beta L_i(q_2).$$

Por tanto las funciones L_i son lineales. Además, si $L_1(q) = L_2(q) = 0$ entonces $q(x_1) = q(x_2) = 0$ y por tanto $q = 0$. Tenemos así que $\mathcal{K} = (K, P, \Sigma)$ es un elemento finito unisolvente. En este caso es fácil ver que la base verifica la δ-propiedad (2.1) puesto que se tiene

$$\begin{aligned} \mathcal{L}_1^1(x_1) &= \frac{(x_1 - x_2)}{(x_1 - x_2)} = 1 \\ \mathcal{L}_1^1(x_2) &= \frac{(x_2 - x_2)}{(x_1 - x_2)} = 0 \\ \mathcal{L}_2^1(x_1) &= \frac{(x_1 - x_1)}{(x_2 - x_1)} = 0 \\ \mathcal{L}_2^1(x_2) &= \frac{(x_2 - x_1)}{(x_2 - x_1)} = 1 \end{aligned}$$

De forma general, tomamos p arbitrario. Tenemos entonces que $P = \mathcal{P}_p(K)$ y la partición será $-1 = x_1 < x_2 < ... < x_{N_P} = 1$.

De forma similar a lo estudiado en el caso $p = 1$ tenemos que L_i son lineales y que $\mathcal{K} = (K, P, \Sigma)$ es un elemento finito unisolvente. Además, esta base verifica la δ-propiedad (2.1) ya que por definición se tiene

$$\mathcal{L}_i^p(x_j) = \delta_{ij}, \quad 1 \leq i, j \leq N_P. \tag{2.3}$$

Ejemplo 2.7. (Terna (K, P, Σ) no unisolvente)

Sea $K = [0, 1]$ y sea $P = \mathcal{P}_1(K)$. Consideramos un conjunto de funciones lineales y continuas $\Sigma = \{\sigma_1, \sigma_2\}$ definidas como

$$\sigma_1(q) = q(0)$$
$$\sigma_2(q) = 2q(0),$$

para cada $q \in P$. Entonces, si la aplicación L es biyectiva, la terna (K, P, Σ) forma un elemento finito (unisolvente).

Pero si $\sigma_1(q) = \sigma_2(q) = 0$ sólo podemos concluir que $q(0) = 0$ y la función no tiene por qué ser la función nula. Por ejemplo, tomando $q(x) = x$ se verifica que $q(0) = 0$ y la función no es la función nula. Por tanto (K, P, Σ) no es unisolvente y por ello no es un elemento finito.

Ejemplo 2.8. (Unisolvencia de elementos jerárquicos)

Sea K un dominio y sea $P = \mathcal{P}_p(K)$ el espacio de polinomios de orden p en K. Consideramos la base $\mathcal{B}^p = \{1, x, x^2, ..., x^p\}$ definida sobre el espacio P. Más adelante veremos que esta base es jerárquica ya que verifica

$$\mathcal{B}^p \subset \mathcal{B}^{p+1}$$

para todo p.

Cada polinomio $q \in P$ puede ser expresado de forma única como combinación lineal de los elementos de la base:

$$q = \sum_{i=1}^{p+1} \beta_i x^{i-1} = \sum_{i=1}^{p+1} L_i(q) x^{i-1}$$

donde β_i son números reales y definimos $L_i(q) = \beta_i$, que son funciones lineales $L_i : P \to \mathbb{R}$ para $i = 1, 2, ..., p + 1$.

Tomando entonces $\Sigma = \{L_1, L_2, ..., L_{N_P}\}$ donde $N_P = p+1$ se tiene que el elemento finito (K, P, Σ) es unisolvente.

Observación 2.9. Se puede demostrar (ver [7]) que, en general, las matrices de elementos finitos asociadas a la base jerárquica anterior están mal condicionadas. Por este motivo, estas bases son poco utilizadas en la práctica y, en su lugar, se utilizan bases jerárquicas cuyos polinomios son ortogonales. Estas bases se estudiarán en la siguientes secciones.

2.2 Mallas de elementos finitos

Definición 2.10. Una *malla*, *mallado*, *triangulación*, $\mathcal{T}_{h,p} = \{K_1, K_2, ..., K_M\}$, de un dominio $\Omega_h \subset \mathbb{R}^n$ acotado con frontera polinómica a trozos es una división geométrica del dominio en un número finito de celdas poligonales K_i cerradas y de forma que no se solapen, donde

$$\Omega_h = \bigcup_{i=1}^{M} K_i. \tag{2.4}$$

Cada celda (a veces llamada, genéricamente, triángulo) K_i, con $1 \leq i \leq M$, puede estar equipada con un orden de polinomios $1 \leq p(K_i) = p_i$, y con un conjunto de grados de libertad propio, Σ_i, de manera que los elementos finitos asociados sean unisolventes.

Observación 2.11. Por simplicidad, gran parte de los resultados teóricos y de las bibliotecas de elementos finitos usados en la práctica suponen que el orden (y grados de libertad) es el mismo en todas las celdas. Pero la posibilidad de particularizar el orden p_i en cada elemento K_i (eventualmente, aumentando el orden en aquellas zonas del dominio donde se estime una mayor dificultad en la aproximación de la solución exacta) es una posibilidad muy interesante, conocida como *p–adaptividad*.

Por otra parte, la posibilidad de refinar el mallado (tomando triángulos más finos donde se estime un mayor error) es más utilizada en la práctica y se denomina *h–adaptividad*. Por este motivo, a la triangulación se la suele denotar simplemente por $\mathcal{T}_h$. Finalmente, la combinación de ambas técnicas se suele denominar *hp–adaptividad*.

A continuación, veremos un par de definiciones usuales referidas a las mallas de elementos finitos.

Definición 2.12. Si se combinan varios tipos de celdas entonces la malla se denomina *híbrida*.

Definición 2.13. Una malla se denomina *admisible* si para cada dos elementos K_i y K_j con $i \neq j$ se tiene sólo uno de los siguientes casos:

- $K_i \cap K_j$ es vacía,
- $K_i \cap K_j$ es un vértice común,
- $K_i \cap K_j$ es un lado común,
- $K_i \cap K_j$ es una cara común.

La admisibilidad del mallado es una condición necesaria para el buen planteamiento de los elementos finitos usuales, aunque no es determinante para elementos finitos discontinuos (ver [8]).

Definición 2.14. Una malla se dice *regular* si "los triángulos no degeneran cuando $h \to 0$". Más concretamente, si existe $\sigma > 0$ tal que

$$\frac{h_K}{\rho_K} \leq \sigma, \quad \forall K \in \mathcal{T}_{h,p}, \quad \forall h > 0,$$

donde $h_K = \text{máx}_{x,y \in K} |x - y|$ es el diámetro de la celda K y ρ_K es el diámetro de la circunferencia inscrita en K.

La regularidad de la malla es una condición que impide que los polígonos contengan ángulos que tienden a cero cuando $h \to 0$. Se utiliza para obtener estimaciones de error en elementos finitos de dimensión mayor o igual que dos (y no tiene sentido en el caso unidimensional).

2.3 Interpolador y conformidad de elementos finitos

Dado un espacio de Hilbert $V(\Omega_h)$, definido sobre un abierto acotado Ω_h y dada una malla $\mathcal{T}_{h,p}$ que recubre Ω_h, cuando tratamos de aproximar una función, $v \in V(\Omega_h)$, mediante el método de los elementos finitos buscamos una función v_h que sea polinómica en cada $K \in \mathcal{T}_{h,p}$ y que, en cierto sentido, interpole a la función original, v. Además, para el desarrollo de los resultados de existencia de la solución v_h y para su convergencia hacia v, es muy interesante el poder asegurar la conformidad, es decir que la aproximación v_h también esté en $V(\Omega_h)$?. En lo que sigue, concretamos estos conceptos.

Sea (K, P, Σ) un elemento finito donde $\Sigma = \{L_1, L_2, ..., L_p\}$ con $L_i : P \to \mathbb{R}$ una función lineal y continua. Suponemos que existe $V(K)$ tal que $P \subset V(K)$ y que cada L_i con $i = 1, ..., p$ se puede extender a $V(K)$, siendo también la extensión, $L_i : V(K) \to \mathbb{R}$, una función lineal y continua sobre $V(K)$.

Definición 2.15. (Interpolador local para un elemento finito)

Sea (K, P, Σ) un elemento finito unisolvente. Sea $\mathcal{B} = \{\theta_1, \theta_2, ..., \theta_{N_P}\}$ la única base de P que satisface la δ-propiedad 2.1. Sea $v \in V \supset P$ una función para la cual todas las formas lineales $L_1, L_2, ..., L_{N_P}$ están definidas. Definimos el *interpolador (local)* como

$$\mathcal{I}_K(v) = \sum_{i=1}^{N_P} L_i(v)\theta_i. \tag{2.5}$$

De la linealidad de las formas L_i se tiene que $\mathcal{I}_K : V \to P$ es lineal.

Proposición 2.16. Sea (K, P, Σ) un elemento finito unisolvente y sea $v \in V \supset P$ una función para la cual las formas lineales L_i están definidas. Entonces

$$L_i(\mathfrak{I}_k(v)) = L_i(v), \;\; 1 \leq i \leq N_P. \tag{2.6}$$

Demostración. Por la definición anterior y la linealidad de las formas L_i se tiene que:

$$L_i(\mathfrak{I}_K(v)) = L_i(\sum_{j=1}^{N_P} L_j(v)\theta_j) = \sum_{j=1}^{N_P} L_j(v)L_i(\theta_j).$$

Así mismo, como la base $\mathcal{B}$ satisface la δ-propiedad, tenemos:

$$\sum_{j=1}^{N_P} L_j(v)L_i(\theta_j) = L_i(v).$$

□

Proposición 2.17. Sea (K, P, Σ) un elemento finito unisolvente. El interpolador de elementos finitos $\mathfrak{I}_K$ es idempotente, esto es, $\mathfrak{I}_K^2 = \mathfrak{I}_K$.

Demostración. De la proposición anterior se tiene que

$$\mathfrak{I}_K(v) = v, \;\; v \in P.$$

Por tanto, para todo $v \in V \supset P$ se tiene

$$\mathfrak{I}_K(\mathfrak{I}_K(v)) = \mathfrak{I}_K(v).$$

□

Observación 2.18. (Interpolación en elementos jerárquicos)

La definición 2.15 no se aplica directamente al caso de la interpolación de elementos jerárquicos ya que los grados de libertad L_i no están definidos para $v \notin P$ (véanse el ejemplo 2.8 y los capítulos siguientes). La definición de operadores locales de interpolación para elementos jerárquicos requiere un análisis matemático más profundo (ver [7], capítulo 3), que utilice la interpolación estándar en elementos de Lagrange y su proyección sobre bases jerárquicas.

Definición 2.19. (Interpolador global)

Sea $\Omega_h \subset \mathbb{R}^n$ un dominio cubierto por una malla de elementos finitos $\mathcal{T}_{h,p}$. Sea $V(\Omega_h)$ un espacio de Hilbert. El interpolador global se define, para cada $v \in V(\Omega_h)$ como una función $\mathfrak{I}(v)$ tal que

$$\mathfrak{I}(v)|_K = \mathfrak{I}_K(v|_K), \quad \forall K \in \mathcal{T}_{h,p}.$$

Usualmente denotaremos por $v_h = \mathcal{I}(v)$. Es natural esperar que si $v \in V(\Omega_h)$, también $v_h \in V(\Omega_h)$, concepto que se conoce como conformidad:

Definición 2.20. (Conformidad de elementos finitos)

Sea $\mathcal{T}_{h,p}$ una malla formada por M elementos finitos unisolventes (K_i, P_i, Σ_i) con $i = 1, ..., M$. Sea $V(\Omega_h)$ un espacio de Hilbert y sean $\mathcal{I}_{K_i} : V(K_i) \to P_i$ los operadores locales de interpolación de cada elemento finito. Decimos que la malla de elementos finitos $\mathcal{T}_{h,p}$ (o el espacio finito-dimensional asociado a esta malla) es *conforme* en el espacio V si existe un subespacio $V^*(\Omega_h) \subset V(\Omega_h)$ que sea denso en V y tal que para cada función $v \in V^*(\Omega_h)$ el correspondiente interpolador global $\mathcal{I}(v)$ esta definido y es un elemento de $V(\Omega_h)$.

Los elementos usados habitualmente son conformes en $H^1(\Omega_h)$. En la siguiente sección estudiaremos la conformidad de algunos espacios de elementos finitos. Para ello, será muy útil el siguiente resultado:

Lema 2.21. (Requisitos de conformidad del espacio H^1)

Sea $\Omega_h \subset \mathbb{R}^n$ un dominio cubierto por una malla de elementos finitos $\mathcal{T}_{h,p}$. Una función $v : \Omega_h \to \mathbb{R}$ pertenece a $H^1(\Omega_h)$ si y sólo si

1. $v|_K \in H^1(K)$ para cada elemento $K \in \mathcal{T}_{h,p}$.
2. Si $e = K_1 \cap K_2$ es una cara c común a dos celdas K_1 y K_2, entonces $v|_{K_1} = v|_{K_2}$ sobre[1] la cara e.

Para la demostración del resultado anterior, véase por ejemplo [7] o [8].

2.4 Espacios de elementos finitos continuos y discontinuos

El método de los elementos finitos consiste en:

a) usar todo lo anterior para construir espacios finito-dimensionales e

b) implementar el método de Galerkin sobre estos espacios.

Nos centramos aquí en la construcción de espacios finito-dimensionales, $V_{h,p}$, lo que habitualmente se lleva a cabo mediante elementos finitos conformes en $H^1(\Omega_h)$. Para ello,

[1] Si $v \in H^1(\Omega)$ y e es un lado común a K_1 y K_2 entonces no está claro el significado de la restricción $v|_e$. El sentido de esta restricción está relacionado con el concepto de traza de v. Para más información ver [1].

se introducirá una malla formada por M elementos finitos unisolventes (K_i, P_i, Σ_i), $i = 1, ..., M$, y un interpolador tal que $v_h = I(v)$ se encuentra en el siguiente espacio

$$V_{h,p} = \{v_h \in C^0(\overline{\Omega}_h) \ / \ v_h|_{K_i} \in \mathcal{P}_{p_i}(K_i), \ \forall i = 1, ..., M\}, \tag{2.7}$$

al que llamaremos *espacio de elementos finitos continuos*.

El lema 2.21 permite que garantizar este espacio es conforme en $H^1(\Omega_h)$, es decir el interpolador global $v_h = I(v)$ está bien definido como función de $H^1(\Omega_h)$. Para comprobarlo, basta tomar en dicho lema $V^*(\Omega_h) = C^0(\Omega_h)$ y $V(\Omega_h) = H^1(\Omega_h)$. Como vimos en el capítulo anterior, $C^0(\Omega)$ es denso en $H^1(\Omega_h)$ (para Ω acotado). Y para cada $v \in V^*(\Omega_h) = C^0(\Omega)$:

1. El interpolador $v_h = \mathcal{I}(v)|_K$ es un polinomio en cada elemento $K \in \mathcal{T}_{h,p}$ y por tanto es de $H^1(K)$.

2. v_h es continuo en Ω_h (por definición), en particular entre las caras de $K \in \mathcal{T}_{h,p}$.

Por tanto, el lema 2.21 garantiza que $\mathcal{I}(v) \in H^1(\Omega_h)$.

Los espacios de elementos finitos habituales son continuos, es decir se ajustan a (2.9), entre ellos los elementos de Lagrange y los elementos jerárquicos de Lobatto que serán estudiados más adelante. En concreto, para la definición de elementos continuos de Lagrange unidimensionales, basta definir (K_i, P_i, Σ_i), $i = 1, ..., M$, como en la definición 2.5 e imponer la continuidad de v_h en cada nodo interior $x_i \in (a, b)$ (lo que supone $M - 1$ restricciones). Como, en el caso 1d, la dimensión de P_i es igual a $p_i + 1$ (donde p_i es el orden de los polinomios), la dimensión de $V_{h,p}$ es igual a

$$\dim(V_{h,p}) = \sum_{i=1}^{M}(p_i + 1) - (M - 1) = \sum_{i=1}^{M} p_i + 1. \tag{2.8}$$

La dimensión de $V_{h,p}$ se puede interpretar como el número global de grados de libertad y el interpolador global puede construirse a partir de combinaciones lineales de bases de $V_{h,p}$, conocidas como bases globales[1]. Con frecuencia[2], se toma $p_i = p$, con $p \in \mathbb{N}$, para todo $i = 1, ..., M$, y entonces $\dim(V_{h,p}) = pM + 1$.

[1] Se trata de una base de $V_{h,p}$ construida a partir de las funciones de forma locales y, eventualmente, de las restricciones de conformidad del interpolador global (continuidad del espacio $V_{h,p}$). En el caso de elementos finitos de Lagrange, éstas son las conocidas "funciones sombrero", que aparecen en la mayor parte de las referencias sobre elementos finitos. Por brevedad, no entraremos en detalles. Ver por ejemplo [5].

[2] Este es el caso de los experimentos numéricos que se muestran en el último capítulo de este trabajo.

Como último comentario: la expresión (2.8) coincide con la dimensión de los sistemas de ecuaciones que se obtendrán al aplicar el método de Galerkin para $V_{h,p}$, que tiende a infinito cuando crecen el número de elementos o el orden de los polinomios. En el caso n-dimensional (o para bases de jerárquicas, por ejemplo de Lobatto) se pueden obtener expresiones similares, que se omiten por brevedad.

De forma alternativa, en el espacio $V_{h,p}$ definido en (2.9) se puede relajar la hipótesis de continuidad, exigiendo solamente $v_h \in L^2(\Omega_h)$, obteniendo el espacio

$$V_{h,p}^{\text{dc}} = \{v_h \in L^2(\Omega_h) \ / \ v_h|_{K_i} \in \mathcal{P}_{p_i}(K_i), \ \forall i = 1, ..., M\}, \tag{2.9}$$

al que llamaremos *espacio de elementos finitos discontinuos*.

Ahora bien, para elementos discontinuos, no se verifica la continuidad entre elementos requerida por el Lema 2.21 y por tanto $\mathcal{I}(v) \notin H^1(\Omega_h)$, es decir, se trata de elementos no conformes en $H^1(\Omega_h)$. Este hecho tiene inconvenientes importantes, entre ellos se dificulta el estudio teórico de la existencia de solución aproximada y de su convergencia. Además, aumenta el número de grados de libertad, por ejemplo en el caso unidimensional, al no existir las $M - 1$ restricciones de continuidad, tenemos:

$$\dim(V_{h,p}^{\text{dc}}) = \sum_{i=1}^{M}(p_i + 1) = \sum_{i=1}^{M} p_i + M.$$

Sin embargo, los elementos finitos discontinuos tienen algunas ventajas: se facilita la aproximación de EDP hiperbólicas (que son difíciles de tratar con elementos continuos, como se puede ver en el ejemplo de las ecuaciones de transporte que aparece en el último capítulo) y abre las puertas a nuevas posibilidades, entre ellas para la computación paralela. Para más detalles, ver [8, 9].

2.5 Implementación del método de los elementos finitos

Concretaremos aquí la forma de llevar a cabo el método de Galerkin para los espacios de elementos finitos definidos en el apartado anterior. En concreto, consideremos un dominio acotado $\Omega \subset \mathbb{R}^n$, una ecuación en derivadas parciales a resolver y un conjunto de condiciones de frontera, cuya formulación variacional viene dada por

$$a(u^* + \overline{u}, v) = f(v), \quad \forall v \in V, \tag{2.10}$$

donde a y f están en las hipótesis del teorema de Lax-Milgram (1.7) y $V \subset H^1(\Omega)$ es un espacio de Hilbert, por ejemplo, $V = H_0^1(\Omega)$. Si las condiciones de frontera originales son no homogéneas, la solución $u = u^* + \overline{u}$ se encuentra en un espacio de funciones afines distinto de V. La función u^* se elige de forma que satisfaga las condiciones de Dirichlet no homogéneas y, realmente, nuestra incógnita es $\overline{u}$, que satisface las condiciones de Dirichlet homogéneas.

2.5.1 Aproximación a formas débiles y discretización

En primer lugar, aproximamos el dominio Ω por otro dominio Ω_h y cubrimos este nuevo dominio por una malla de elementos finitos $\mathcal{T}_{h,p}$.

Aproximamos el espacio V por un subespacio $V_{h,p}$, como el que viene dado en (2.9). Eventualmente, aproximamos la forma bilineal a por otra forma $a_{h,p}$ y la forma f por $f_{h,p}$, de forma que ahora la integración vaya sobre Ω_h y $\partial\Omega_h$.

Así, la solución $u_{h,p}$ es buscada de la forma $u_{h,p} = u^*_{h,p} + \overline{u}_{h,p}$ con $\overline{u}_{h,p} \in V_{h,p}$, satisfaciendo [1]

$$a_{h,p}(u^*_{h,p} + \overline{u}_{h,p}, v_{h,p}) = f_{h,p}(v_{h,p}), \tag{2.11}$$

para todo $v_{h,p} \in V_{h,p}$. Expresamos la función $\overline{u}_{h,p}$ como una combinación lineal de las funciones base globales v_i del espacio $V_{h,p}$

$$u_{h,p}(x) = u^*_{h,p}(x) + \overline{u}_{h,p}(x) = u^*_{h,p} + \sum_{j=1}^{N} y_j v_j(x), \;\; y_j \in \mathbb{R}. \tag{2.12}$$

Sustituyendo esta última equación en (2.11) y eligiendo $v_{h,p} = v_i$, con $i = 1, ..., N$ tenemos

$$a_{h,p}(u^*_{h,p} + \sum_{j=1}^{N} y_j v_j(x), v_i) = f_{h,p}(v_i), \;\; i = 1, ..., N. \tag{2.13}$$

Como $a_{h,p}$ es una forma bilineal, se tiene:

$$\sum_{j=1}^{N} a_{h,p}(u^*_{h,p}, v_i) + a_{h,p}(v_j(x), v_i) y_j = f_{h,p}(v_i), \;\; i = 1, ..., N.$$

[1] Atención: si $V_{h,p}$ fuera un espacio no conforme en $H^1(\Omega_h)$, podría no tener sentido la formulación variacional discreta 2.11. De aquí provienen, en parte, las dificulades teóricas en los elementos finitos no conformes.

Esto es,

$$\sum_{j=1}^{N} a_{h,p}(v_j(x), v_i) y_j = f_{h,p}(v_i) - a_{h,p}(u^*_{h,p}, v_i), \;\; i = 1, ..., N. \tag{2.14}$$

Esto lo podemos ver como un sistema de ecuaciones lineales $SY = F$, donde

$$\begin{aligned} S_{ij} &= a_{h,p}(v_j(x), v_i), \\ Y_j &= y_j, \\ F_i &= f_{h,p}(v_i) - a_{h,p}(u^*_{h,p}, v_i). \end{aligned}$$

Por último, hay que resolver el sistema de ecuaciones anterior para la matriz de coeficientes Y. Como se comentó al hablar del método de Galerkin, este sistema está bien planteado (por ejemplo, bajo las condiciones de Lax-Milgram). Además, en este momento, se puede apreciar una de las ventajas del método de los elementos finitos: la matriz asociada al sistema de ecuaciones es una matriz hueca (formada por un gran número de ceros). Esto se debe a que las funciones base globales están definidas a partir de las funciones de forma locales de manera que aquéllas (y sus derivadas) tienen soporte disjunto.

2.6 Dominios y aplicaciones de referencia

A la hora de la implementación de los elementos finitos, es necesario calcular un gran número de integrales, sobre cada elemento de una malla, para así poder evaluar los elementos S_{ij} y F_i correspondientes a la matriz y al segundo miembro del sistema de ecuaciones dado en el apartado anterior.

En las aproximaciones lineales a trozos los grados de libertad están asociados con los valores de la solución en los vértices de la malla, no es complicado evaluar directamente estas integrales sobre los elementos la malla.

Sin embargo, para elementos de orden alto, especialmente en dimensión mayor que uno, la situación es mucho más complicada.

Por ello, para discretizaciones de elementos finitos de orden alto, las celdas de la malla $K_i \in \mathcal{T}_{h,p}$ suelen ser transformadas a un dominio de referencia $\widehat{K}$ mediante aplicaciones de referencia biyectivas

$$F_{K_i} : \widehat{K} \to K_i, \tag{2.15}$$

de forma que todas las integrales pueden realizarse, en el elemento de referencia y los elementos S_{ij} y F_i pueden calcularse a través de un cambio de variables.

Observación 2.22. En este trabajo, para el caso unidimensional, tomaremos como dominio de referencia el intervalo $\widehat{K} = [-1, 1]$.

CAPITULO

3

Polinomios ortogonales y elementos finitos jerárquicos

Una vez analizada la teoría abstracta del método de los elementos finitos y estudiados los elementos finitos nodales, podemos definir algunas familias de elementos finitos jerárquicos que tendrán buenas propiedades de cara a la resolución del sistema de ecuaciones subyacente a este método numérico.

3.1 Elementos finitos jerárquicos

Sea un dominio $K \subset \mathbb{R}$. Para cada $p = 0, 1, 2, ...$ consideremos el espacio P de polinomios de orden p. Denotamos como N_P a la dimensión de P y

$$\mathcal{B}^p = \{\theta_1, \theta_2, ..., \theta_{N_P}\},$$

una base de P.

Definición 3.1. Decimos que una familia de bases $\{\mathcal{B}^p\}_{p\geq 0}$ es *jerárquica* si

$$\mathcal{B}^p \subset \mathcal{B}^{p+1}. \tag{3.1}$$

Así mismo, decimos que una base es *jerárquica* si pertenece a una familia de bases jerárquicas.

Definición 3.2. Llamamos *elemento finito jerárquico* a un elemento finito (K, P, Σ) cuyos grados de libertad $\Sigma = \{L_1, ..., L_{N_P}\}$ vienen definidos por

$$\begin{aligned} L_i : P &\longrightarrow \mathbb{R} \\ q &\longmapsto \beta_i, \end{aligned}$$

con $i = 1, 2, ..., N_P$ y donde $\beta_i \in \mathbb{R}$ son las únicas constantes que verifican

$$q = \sum_{i=1}^{N_P} \beta_i \theta_i = \sum_{i=1}^{N_P} L_i(q)\theta_i, \tag{3.2}$$

siendo $\mathcal{B}^P = \{\theta_1, ..., \theta_{N_P}\}$ una base jerárquica de P.

Observación 3.3. En muchas referencias bibliográficas, a estos elementos también se les llama *elementos finitos espectrales*, aunque este nombre puede dar pie a confusiones porque con frecuencia se asocia específicamente a *métodos espectrales* relacionados con el uso de series de Fourier para la construcción de los espacios finito-dimensionales.

3.2 Polinomios ortogonales

Definición 3.4. Sea un espacio de Hilbert H, dotado de un producto escalar $(\cdot, \cdot)$, y sean $f, g \in H$. Entonces:

- Decimos que $f, g \in H$ son *ortogonales* si $(f, g) = 0$.
- Decimos que f, g son *ortonormales* si son ortogonales y además $||f|| = ||g|| = 1$, donde $|| \cdot || = +\sqrt{(\cdot, \cdot)}$ es la norma asociada al producto escalar.
- Una base B de H, se dice *ortogonal* si todos los elementos de B son ortogonales dos a dos, esto es,
$$(\varphi, \phi) = 0, \quad \forall \varphi, \phi \in B,\ \varphi \neq \phi. \tag{3.3}$$
De manera análoga se puede definir una base ortonormal.

Consideremos el espacio $\mathbb{P}_n[x]$, que es un subespacio finito-dimensional de $L^2(a, b)$. Por tanto, $\mathbb{P}_n[x]$ es un espacio de Hilbert para el producto escalar de $L^2(a, b)$. Una base $\{\phi_0, \phi_1, ..., \phi_n\}$ de $\mathbb{P}_n[x]$ es ortogonal en $L^2(a, b)$ si $\int_a^b \phi_i \phi_j = 0, \ \ \forall i \neq j$.

Proposición 3.5. Dado $[a, b] \subset \mathbb{R}$ se tiene que para todo $n \geq 0$ existen bases de $\mathbb{P}_n[x]$ que son ortonormales en $L^2(a, b)$.

Demostración. Basta aplicar el método de Gram-Schmidt al espacio finito-dimensional $\mathbb{P}_n[x]$. □

Lo anterior se generaliza fácilmente a polinomios de varias variables.

3.2.1 Polinomios de Legendre

Los polinomios de Legendre forman una base ortonormal del espacio $L^2(I)$, con $I = [-1, 1]$. En principio, estos polinomios fueron construidos por el método de Gram-Schmidt y posteriormente se encontraron muchas propiedades interesantes.

Los polinomios de Legendre vienen definidos por la siguiente regla de recurrencia:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_k(x) &= \frac{2k-1}{k} x L_{k-1}(x) - \frac{k-1}{k} L_{k-2}(x), \quad k = 2, 3, ... \end{aligned}$$

Su ortogonalidad en $L^2(I)$ se puede verificar por recurrencia. Además, verifican una serie de proposiciones que son bien conocidas (ver [7]), entre las cuales tenemos:

Proposición 3.6.

1. Los polinomios de Legendre pueden ser definidos de forma análoga por:

$$L_k(x) = \frac{1}{2^k k!} \frac{d^k}{dx^k}(x^2 - 1)^k, \quad k = 0, 1, 2, ...$$

2. Los polinomios de Legendre satisfacen la ecuación diferencial de Legendre,

$$(1 - x^2)\frac{d^2y}{dx^2} - 2x\frac{dy}{dx} + k(k+1)y = 0.$$

3. La ortogonalidad de estos polinomios viene definida por

$$\int_{-1}^{1} L_k(x) L_m(x) \mathrm{d}x = \begin{cases} \frac{2}{2k+1}, & \text{si } k = m, \\ 0, & \text{c.c.} \end{cases} \tag{3.4}$$

Además es fácil comprobar que se verifica la siguiente proposición:

Proposición 3.7. Los polinomios de Legendre satisfacen

$$\begin{aligned} L_n(1) &= 1, \\ L_n(-1) &= (-1)^n, \quad n \geq 0. \end{aligned}$$

Demostración. Lo demostraremos por inducción:

- Para $n = 0$ se tiene $L_0(1) = L_0(-1) = 1$.
- Para $n = 1$ se tiene $L_1(1) = 1$ y $L_1(-1) = -1$.

Suponemos ahora que se verifica para todo $n < k$, entonces para $n = k$:

$$L_k(1) = \frac{2k-1}{k} - \frac{k-1}{k} = \frac{2k-k-1+1}{k} = \frac{k}{k} = 1,$$
$$L_k(-1) = \begin{cases} \frac{2k-1}{k} - \frac{k-1}{k} = \frac{2k-k-1+1}{k} = \frac{k}{k} = 1, & \text{si } k \text{ es par,} \\ -\frac{2k-1}{k} + \frac{k-1}{k} = \frac{-2k+k-1+1}{k} = \frac{-k}{k} = -1, & \text{si } k \text{ es impar.} \end{cases}$$

□

Vamos a escribir algunos polinomios de Legendre:

$$\begin{aligned} L_0(x) &= 1, \\ L_1(x) &= x, \\ L_2(x) &= \frac{3}{2}x^2 - \frac{1}{2}, \\ L_3(x) &= \frac{1}{2}x(5x^2 - 3), \\ L_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3), \\ L_5(x) &= \frac{1}{8}x(63x^4 - 70x^2 + 15), \\ L_6(x) &= \frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5), \\ L_7(x) &= \frac{1}{16}x(429x^6 - 693x^4 + 315x^2 - 35). \end{aligned}$$

Podemos ver estos polinomios dibujados en la figura 3.1.

3.2.2 **Polinomios de Lobatto**

Los polinomios de Lobatto vienen definidos a partir de los polinomios de Legendre y se introducen aquí porque, como veremos en el siguiente capítulo, dan lugar a una matriz de rígidez con buenas propiedades (ver Observación 4.5).

Podemos definir los polinomios de Lobatto de la siguiente forma:

$$\begin{aligned} l_0(x) &= \frac{1-x}{2}, \\ l_1(x) &= \frac{x+1}{2}, \\ l_k(x) &= \frac{1}{||L_{k-1}||_2} \int_{-1}^{x} L_{k-1}(\xi)\mathrm{d}\xi, \;\; 2 \leq k, \end{aligned}$$

donde $||L_{k-1}||_2 = \sqrt{\frac{2}{2k-1}}$ (ver 3.4).

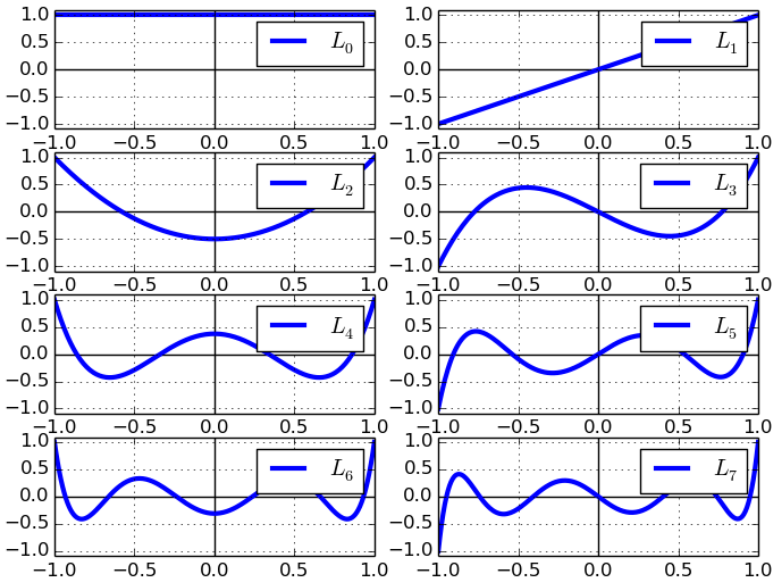


Figura 3.1: Polinomios de Legendre.

El interés de los polinomios de Lobatto proviene del hecho de que sus derivadas para $k \geq 2$ son proporcionales a los polinomios de Legendre:

$$l'_k(x) = \frac{L_{k-1}(x)}{||L_{k-1}||_2}. \tag{3.5}$$

De hecho, se verifica que para $k \geq 2$ estos polinomios son ortonormales:

$$||l'_k(x)||_2 = \frac{||L_{k-1}||_2}{||L_{k-1}||_2} = 1. \tag{3.6}$$

Además, se tiene la siguiente propiedad:

Proposición 3.8. Se verifica que

$$\begin{aligned} l_k(-1) &= 0, \\ l_k(1) &= 0, \quad k = 2, 3, ... \end{aligned}$$

Demostración. Sea $k \geq 2$ entonces

$$\begin{aligned} l_k(-1) &= \frac{1}{||L_{k-1}||_2} \int_{-1}^{-1} L_{k-1}(\xi) \mathrm{d}\xi = 0, \\ l_k(1) &= \frac{1}{||L_{k-1}||_2} \int_{-1}^{1} L_{k-1}(\xi) \mathrm{d}\xi = \frac{1}{||L_{k-1}||_2} \int_{-1}^{1} L_{k-1} L_0(\xi) \mathrm{d}\xi = 0. \end{aligned}$$

□

Vamos a escribir algunos polinomios de Lobatto:

$$
\begin{aligned}
l_2(x) &= \frac{1}{2}\sqrt{\frac{3}{2}}(x^2-1), \\
l_3(x) &= \frac{1}{2}\sqrt{\frac{5}{2}}(x^2-1)x, \\
l_4(x) &= \frac{1}{8}\sqrt{\frac{7}{2}}(x^2-1)(5x^2-1), \\
l_5(x) &= \frac{1}{8}\sqrt{\frac{9}{2}}(x^2-1)(7x^2-3)x, \\
l_6(x) &= \frac{1}{16}\sqrt{\frac{11}{2}}(x^2-1)(21x^4-14x^2+1), \\
l_7(x) &= \frac{1}{16}\sqrt{\frac{13}{2}}(x^2-1)(33x^4-30x^2+5)x,
\end{aligned}
$$

Podemos ver los polinomios de Lobatto dibujados en la siguiente figura:

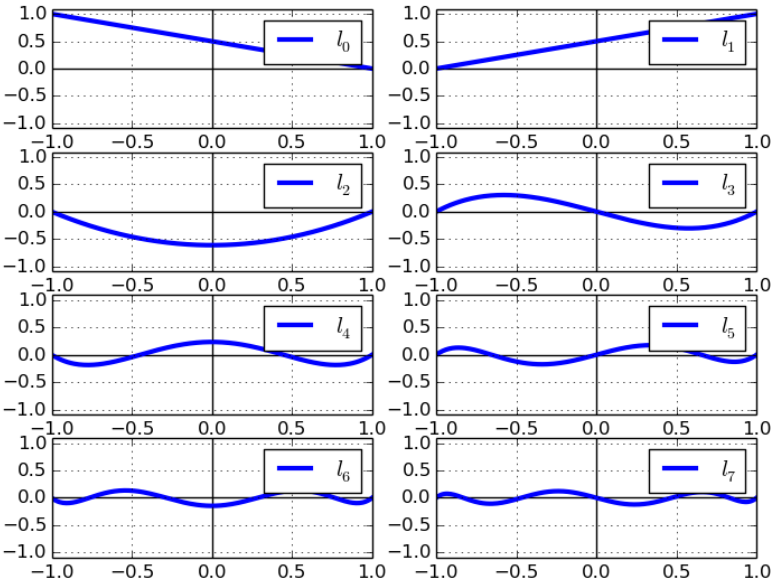


Figura 3.2: Polinomios de Lobatto.

CAPITULO

4

Implementación de elementos finitos con bases jerárquicas

En este capítulo vamos a desarrollar las nociones planteadas en los capítulos precedentes estudiando la implementación en la práctica de elementos finitos jerárquicos. Para ello plantearemos un problema modelo unidimensional y veremos cuales son los pasos a seguir para su resolución numérica utilizando elementos finitos continuos definidos mediante bases de Lobatto.

La elección de este tipo de polinomios está directamente determinada por la EDP planteada. Como veremos, nos permitirá resolver el problema mediante elementos finitos continuos con matrices de excelentes propiedades (un óptimo número de condición y, de hecho, prácticamente diagonales), aunque estas afirmaciones no se extenderían directamente a otro tipo de EDP. Por otra parte, su extensión al caso bidimensional (o n–dimensional) es sencilla para mallas definidas como producto cartesiano de mallas 1d. Su extensión a mallas triangulares también es posible, aunque se requiere un proceso algo más elaborado (ver Solin [7], capítulo 2).

4.1 El problema continuo y discreto

Consideremos el intervalo $I = (a, b) \subset \mathbb{R}$ y una función $f \in L^2(I)$. Queremos resolver la ecuación de Poisson

$$-u''(x) = f(x) \tag{4.1}$$

en I, con condiciones de contorno de tipo Dirichlet

$$u(a) = g_a, \tag{4.2}$$

$$u(b) = g_b. \tag{4.3}$$

En este apartado, para estudiar condiciones de contorno de tipo Dirichlet no homogéneas utilizamos la siguiente notación:

$$u(x) = u^*(x) + \overline{u}(x), \tag{4.4}$$

de forma que $u^* \in H^1(a,b)$ satisfaga las condiciones (4.2) y (4.3),

$$u^*(a) = g_a, \tag{4.5}$$

$$u^*(b) = g_b, \tag{4.6}$$

y que la función $\overline{u}$ verifique las condiciones de frontera homogéneas

$$\overline{u}(a) = \overline{u}(b) = 0. \tag{4.7}$$

Nuestra incógnita ahora es la función $\overline{u} \in V = H_0^1$ de forma que satisfaga la formulación variacional

$$\int_a^b [u^*(x) + \overline{u}(x)]' v'(x) \mathrm{d}x = \int_a^b f(x)v(x)\mathrm{d}x, \;\; v \in V. \tag{4.8}$$

Esto es equivalente a

$$\int_a^b [\overline{u}]'(x) v'(x) \mathrm{d}x = \int_a^b [f(x)v(x) - [u^*]x'(x)v'(x)]\mathrm{d}x, \;\; v \in V. \tag{4.9}$$

4.1.1 **Discretización de (4.9)**

Para hacer la discretización, necesitamos una malla $\mathcal{T}_{h,p} = \{K_1, ..., K_M\}$ junto a espacios de polinomios $\{P_1, ..., P_M\}$ determinados por órdenes $p_1, p_2, ..., p_M$, de forma que para cada K_i y K_j con $i \neq j$, no tiene por qué ser $p_i = p_j$. En el caso de elementos finitos de Lagrange, los grados de libertad vendrán determinados por los valores de cada polinomio en una serie de nodos (como se especificó en el capítulo 2), mientras que para elementos finitos nodales los grados de libertad vienen dados por las coordenadas de un polinomio en la base jerárquica correspondiente (en este caso, en la base de Lobatto), como se especificó en el capítulo anterior.

Obsérvese que, los dos primeros elementos de las bases jerárquicas de Lobatto, l_0 y l_1, coinciden (en el intervalo de referencia $[-1,1]$) con la base de Lagrange de orden $p=1$. Esto se puede interpretar en el siguiente sentido: *los elementos con bases de Lobatto de orden $p=1$ son nodales y dan pie a la elaboración de espacios de elementos finitos continuos*. En general, la continuidad de este tipo de elementos se puede extender a orden $p>1$, pues (debido a su carácter jerárquico) todas ellas contienen a l_0 y l_1. Una observación interesante, es que las bases de Legendre no verifican esta propiedad y por ello, es difícil usarlas para definir espacios de elementos finitos continuos.

Tomamos como dominio de referencia $\widehat{K}=(-1,1)$ y definimos para cada elemento $K_i=(x_i,x_{i+1}),\ i=1,2,...,M$ una aplicación de referencia

$$\begin{aligned} F_{K_i}:\widehat{K} &\longrightarrow K_i, \\ \hat{x} &\longmapsto F_{K_i}(\hat{x})=c_1^{(i)}+c_2^{(i)}\hat{x}, \end{aligned} \tag{4.10}$$

que verifique $F_{K_i}(-1)=x_i$ y $F_{K_i}(1)=x_{i+1}$. Por tanto,

$$\begin{aligned} c_1^{(i)} &= \frac{x_i+x_{i+1}}{2}, \\ c_2^{(i)} &= \frac{x_{i+1}-x_i}{2}. \end{aligned}$$

De esta forma, el espacio V se aproxima por el subespacio

$$V_{h,p}=\{v\in V : v|_{K_i}\circ F_{K_i}\in P^{p_i}(\widehat{K}) \text{ para todo } i=1,2,...,M\}, \tag{4.11}$$

que está formado por funciones continuas debido al lema (2.21). Entonces, teniendo en cuenta la continuidad de $V_{h,p}$, haciendo un recuento de dimensiones y grados de libertad y debido a la continuidad del espacio de elementos finitos resultante, no es difícil llegar (como en el capítulo 2) a que:

$$N=\dim(V_{h,p})=M-1+\sum_{i=1}^{M}(p_i+1)=1+\sum_{i=1}^{M}p_i. \tag{4.12}$$

Tenemos así que el problema discreto asociado a (4.9) es el siguiente:

$$\int_a^b [\overline{u}_{h,p}]'(x)v'_{h,p}(x)\mathrm{d}x=\int_a^b [f(x)v_{h,p}(x)-[u^*_{h,p}]'(x)v'_{h,p}(x)]\mathrm{d}x, \tag{4.13}$$

para todo $v_{h,p}\in V_{h,p}$.

Utilizando el método de Galerkin de la forma especificada en el capítulo 2, el problema discreto (4.13) puede transformarse en un sistema de ecuaciones de dimensión N cuya

solución determinará, en cada celda K, los valores de la solución como combinación lineal de las funciones de forma.

Observación 4.1. Para la construcción de este sistema de ecuaciones, suele utilizarse un algoritmo estándar en la teoría de elementos finitos, que consiste en:

a) Realizar un bucle por cada celda, K_i.

b) En K_i, realizar un doble bucle por sus funciones de forma locales j, k.

c) Calcular, en la celda K_i, la integral de estas funciones de forma y añadirla a la posición adecuada [1] de la matriz global de elementos finitos.

Esta es la técnica que se ha utilizado en la biblioteca de elementos finitos que ha sido desarrollada como parte de este trabajo, y que se presenta en la siguiente sección.

4.1.2 Elección de la función de levantamiento $\mathbf{u}^*$

Es natural pedir que la solución $u_{h,p} = u^*_{h,p} + \overline{u}_{h,p}$ sea un polinomio de orden p_i para cada elemento K_i, $i = 1, 2, ..., M$. Como $\overline{u}_{h,p} \in V_{h,p}$ (y por tanto se tiene que es un polinomio de orden p_i en K_i), para que se verifique lo anterior tiene que ser $u^*_{h,p}$ un polinomio de orden p_i en K_i. En la práctica elegimos $u^*_{h,p}$ lo más sencillo posible, por ejemplo como una función continua y lineal a trozos de forma que se anule en los elementos interiores y cuyo valor en la frontera coincida con las condiciones de contorno.

En este caso, la condición de contorno suele ser implementada en la práctica a través de una técnica[2] conocida como bloqueo de los grados de libertad de las fronteras Dirichlet. Ésta consiste en ignorar las condiciones de contorno a la hora de la construcción de los sistemas lineales para, posteriormente, modificar el sistema de ecuaciones resultante (concretamente las posiciones correspondientes a los grados de libertad correspondientes a las fronteras con condiciones Dirichlet) de forma que los valores en los nodos frontera coincidan con los especificados por las condiciones de contorno.

4.2 Transformación al dominio de referencia $\widehat{\mathbf{K}}$

Lo primero que debemos hacer a la hora de trabajar con nuestro problema discreto es pasar de cada elemento K_i de la malla $\mathcal{T}_{h,p}$ al dominio de referencia $\widehat{K}$. Esto hace que no sea

[1] Utilizando información preestablecida "de conectividad" que enlace los índices locales con las filas y columnas de la matriz global.

[2] Que ha sido la técnica utilizada para fijar las condiciones de contorno en los experimentos numéricos que se muestran en la siguiente sección.

necesario hacer el mismo cálculo para cada elemento de la malla, sólo hay que hacerlo en el elemento de referencia, siempre y cuando exista una función biyectiva que transforme un elemento en otro. En la práctica, se supondrá una transformación afín, como la dada en (4.10).

Para cada elemento de la malla, K_i, definimos $\overline{u}_{h,p}^{(i)}$ como la transformación de $\overline{u}_{h,p}$ al dominio de referencia $\widehat{K}$, es decir,

$$\overline{u}_{h,p}^{(i)}(\hat{x}) = \overline{u}_{h,p}(F_{K_i}(\hat{x})) = \overline{u}_{h,p}(x)|_{K_i}. \tag{4.14}$$

Si derivamos la función $\overline{u}_{h,p}^{(i)}$ obtenemos que

$$[\overline{u}_{h,p}^{(i)}]'(\hat{x}) = [\overline{u}_{h,p} \circ F_{K_i}]'(\hat{x}) = \overline{u}'_{h,p}(x)|_{K_i} J_{K_i}, \tag{4.15}$$

donde J_{K_i} representa el jacobiano de la función F_{K_i}. De manera análoga, definimos $u_{h,p}^{*(i)}$, $v_{h,p}^{(i)}$ y $f^{(i)}$ como las transformaciones de $u_{h,p}^*$, $v_{h,p}$ y f al dominio de referencia $\widehat{K}$.

Haciendo un cambio de variables en cada uno de los miembros de (4.13) tenemos que

$$\int_{K_i} [\overline{u}_{h,p}]'(x) v'_{h,p}(x) \mathrm{d}x = \int_{\widehat{K}} \frac{1}{J_{K_i}} [\overline{u}_{h,p}^{(i)}]'(\hat{x}) [v_{h,p}^{(i)}]'(\hat{x}) \mathrm{d}\hat{x}, \tag{4.16}$$

y

$$\int_{K_i} \left(f(x) v_{h,p}(x) - [u_{h,p}^*]'(x) v'_{h,p}(x) \right) \mathrm{d}x = \\ = \int_{\widehat{K}} \left(J_{K_i} f^{(i)}(\hat{x}) v_{h,p}^{(i)}(\hat{x}) - \frac{1}{J_{K_i}} [u_{h,p}^{*(i)}]'(\hat{x}) [v_{h,p}^{(i)}]'(\hat{x}) \right) \mathrm{d}\hat{x}. \tag{4.17}$$

Por tanto, transformamos la ecuación local (4.13) en la siguiente ecuación en el triángulo de referencia $\widehat{K}$:

$$\int_{\widehat{K}} \frac{1}{J_{K_i}} [\overline{u}_{h,p}^{(i)}]'(\hat{x}) [v_{h,p}^{(i)}]'(\hat{x}) \mathrm{d}\hat{x} = \int_{\widehat{K}} \left(J_{K_i} f^{(i)}(\hat{x}) v_{h,p}^{(i)}(\hat{x}) - \frac{1}{J_{K_i}} [u_{h,p}^{*(i)}]'(\hat{x}) [v_{h,p}^{(i)}]'(\hat{x}) \right) \mathrm{d}\hat{x}.$$

Esta ecuación permite, con el cambio de variables $x = F(\hat{x})$, aplicar en cada elemento K_i el algoritmo comentado en la observación 4.1.

4.3 Funciones de forma de alto orden

4.3.1 Funciones de forma de alto orden nodales

Sea K_i un elemento con un cierto orden de polinomios p_i. Para simplificar los cálculos elegimos los nodos que forman la partición del dominio de referencia de forma equiespaciada, para ello definimos

$$x_{j+1}^{(i)} = -1 + \frac{2j}{p_i} \in \widehat{K}, \;\; j = 0, 1, ..., p_i. \tag{4.18}$$

Utilizando los polinomios de interpolación de Lagrange (ver Definición 2.5) y la δ-propiedad (2.1) obtenemos $p_i + 1$ funciones nodales de orden p_i de la forma

$$\theta_l(\hat{x}) = \frac{(\hat{x} - x_1^{(i)})(\hat{x} - x_2^{(i)})...(\hat{x} - x_{l-1}^{(i)})(\hat{x} - x_{l+1}^{(i)})...(\hat{x} - x_{p_i+1}^{(i)})}{(x_l^{(i)} - x_1^{(i)})(x_l^{(i)} - x_2^{(i)})...(x_l^{(i)} - x_{l-1}^{(i)})(x_l^{(i)} - x_{l+1}^{(i)})...(x_l^{(i)} - x_{p_i+1}^{(i)})}. \quad (4.19)$$

Para el caso en el que $p_i = 1$ consideramos los puntos $x_1^{(i)} = -1$ y $x_2^{(i)} = 1$ y, por tanto, las dos funciones formas que consideramos vienen definidas por

$$\theta_1 = \frac{1 - \hat{x}}{2},$$
$$\theta_2 = \frac{\hat{x} + 1}{2}.$$

Estas funciones tienen la ventaja de que los coeficientes que se obtienen resolviendo el problema discreto, son precisamente el valor de la solución aproximada $u_{h,p}$ en los nodos $x_j^{(i)}$. Sin embargo, estas funciones no son jerárquicas y, por tanto, para aumentar el orden de los polinomios deberíamos volver a calcular todas la funciones de forma. Además, si consideramos grandes dimensiones espaciales sería complicado combinar elementos nodales con el hecho de que en la malla haya distintos órdenes de polinomios. Esta afirmación es especialmente válida para elementos finitos continuos en problemas de dimensión mayor que uno, donde para asegurar la continuidad entre dos celdas K_1 y K_2 es necesario que exista (dependiendo del orden de los polinomios) un cierto número de nodos en la frontera común de K_1 y K_2, sobre los que la solución sea coincidente.

Observación 4.2. Sin embargo, para bases de Lobatto (que tratamos en la siguiente sección), la continuidad entre elementos viene dada exclusivamente por las dos primeras funciones base, mientras que el resto de ellas viene dada como funciones *burbuja* (que se anulan en la frontera de cada celda K_i, definidas en la sección 4.4.2). De esta forma, es fácil aumentar el orden polinómico en un elemento finito (y poco costoso, debido al carácter jerárquico de las bases). En este sentido, las bases jerárquicas facilitan la p–adaptividad o la hp–adaptividad.

4.3.2 Funciones de forma de alto orden jerárquicas

Definición 4.3. Sea $\{\theta_1, \theta_2, ..., \theta_{p+1}\}$ una base del espacio de polinomios $\mathcal{P}^p(\widehat{K})$. Entonces *la matriz de rígidez en el elemento de referencia* de orden p correspondiente al problema (4.1) es una matriz de dimensión $(p + 1) \times (p + 1)$ de la forma

$$\hat{S} = \{\hat{s}_{i,j}\}_{i,j=1}^{p+1}, \quad \text{donde} \quad \hat{s}_{ij} = \int_{\widehat{K}} \theta_i'(\hat{x})\theta_j'(\hat{x})\mathrm{d}\hat{x}. \quad (4.20)$$

Dentro de las funciones de forma jerárquicas unas de las más usuales son las funciones forma de Lobatto que ya fueron introducidas en la sección (3.2.2). Estos polinomios tienen unas excelentes propiedades para la construcción de matrices de rigidez, ya que sus derivadas son los polinomios de Legendre. De esta forma tenemos la siguiente propiedad:

$$\int_{-1}^{1} l'_{i-1}(\hat{x}) l'_{j-1}(\hat{x}) \mathrm{d}\hat{x} = 0, \quad \text{siempre que } i > 2 \text{ o } j > 2,\ i \neq j. \tag{4.21}$$

Observación 4.4. Ya hemos visto en apartados anteriores que las integrales de rigidez de la formulación variacional dadas mediante una transformación lineal del elemento $K_i \in \mathcal{T}_{h,p}$ al dominio de referencia $\widehat{K}$ mantienen su forma original salvo multiplicación por una constante (el inverso del Jacobiano de la aplicación afín F_{K_i}). Por ello, todas las integrales necesarias para la matriz de rigidez global, se pueden obtener de la matriz de rígidez en el elemento de referencia. Esto reduce de forma notable el coste computacional.

Sin embargo, hay que tener en cuenta que esto sólo es cierto cuanto el operador de la formulación variacional no depende explícitamente del espacio. Es decir, si en lugar de tener $L(u) = u''$ tuviésemos $\widetilde{L}(u) = ((1+x)u')'$ no se podría aplicar esta propiedad.

Observación 4.5. La relación (4.21) implica que la matriz $\hat{S}$ de orden p para las funciones forma de Lobatto $l_0, l_1, ..., l_p$ viene dada de la forma

$$\hat{S} = \begin{bmatrix} \hat{s}_{11} & \hat{s}_{12} & 0 & 0 & ... & 0 \\ \hat{s}_{21} & \hat{s}_{22} & 0 & 0 & ... & 0 \\ 0 & 0 & \hat{s}_{33} & 0 & ... & 0 \\ 0 & 0 & 0 & \ddots & & 0 \\ \vdots & & & & & 0 \\ 0 & ... & & & & \hat{s}_{p+1,p+1} \end{bmatrix}.$$

Podemos ver que es una matriz casi diagonal pues sólo fallan las dos primeras filas, las cuales corresponden al producto de las funciones l_0 y l_1. Esto hace que la resolución del sistema de ecuaciones que nos queda sea mucho más sencillo computacionalmente.

4.3.3 El número de condición en bases jerárquicas

A continuación, estudiamos el número de condición de esta matriz de rigidez. Antes de ello, algunas definiciones y propiedades relacionadas con este concepto (estrechamente relacionado con la calidad en la resolución numérica de los sistemas de ecuaciones). La demostración de estos resultados se puede ver en [10].

Definición 4.6. Sea M una matriz regular de dimensión $n \times n$. Llamamos *número de condición* de la matriz M al producto

$$\chi(M) = ||M||||M^{-1}||, \tag{4.22}$$

donde $||.||$ es la norma matricial.

Proposición 4.7. Se tiene que

$$\chi(M) \geq 1, \tag{4.23}$$

para cualquier norma matricial.

Demostración.

$$\chi(M) = ||M||||M^{-1}|| \geq ||MM^{-1}|| = ||I|| = 1, \tag{4.24}$$

donde I es la matriz identidad. □

Definición 4.8. La norma matricial de la definición anterior puede escogerse de múltiples formas. Dos de las más comunes son la *norma euclídea*

$$||M|| = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} m_{i,j}^2}, \tag{4.25}$$

y la *norma espectral*

$$||M|| = \sqrt{\rho(MM^T)}, \tag{4.26}$$

donde $\rho(MM^T)$ es el *radio espectral* de la matriz simétrica y definida positiva MM^T. Usando esta última podemos obtener el *número de condición espectral*

$$\chi^*(M) = \sqrt{\frac{\lambda_n(MM^T)}{\lambda_1(MM^T)}}, \tag{4.27}$$

donde $\lambda_n(MM^T)$ y $\lambda_1(MM^T)$ son el mayor y el menor de los autovalores de MM^T.

Proposición 4.9. Se verifica que

$$\chi^*(M) \leq \chi(M), \tag{4.28}$$

para cualquier número de condición $\chi(M)$.

En [7] podemos ver una comparación del número de condición de la matriz de rigidez asociada a las bases tanto de Lagrange como de Lobatto. En concreto, basándonos en esta referencia, podemos afirmar lo siguiente:

- Para polinomios de orden $p = 2$, el número de condición de ambas matrices es 1.
- Sin embargo, conforme p aumenta:
  1. El número de condición de la matriz asociada a las *bases de Lagrange* también aumenta, llegando a alcanzar *valores superiores a* 10000 para $p = 10$
  2. Mientras que para la matriz asociada a las *bases de Lobatto* el número de condición *siempre es* 1*, es decir el número de condición es óptimo*. Esta validez sólo se verifica para funciones en dimensión 1, aunque en dimensiones superiores el número de condición es también muy adecuado.

Por supuesto, lo anterior se aplica solamente a matrices de rigidez, y el número de condición de otro tipo de matrices deja de ser óptimo para las bases de Lobatto. Por ejemplo, para matrices de masa, definidas en el intervalo de referencia como integrales de las funciones de forma, es decir

$$\hat{M} = \{\hat{m}_{i,j}\}_{i,j=1}^{p+1}, \quad \text{donde} \quad \hat{m}_{ij} = \int_{\widehat{K}} \theta_i(\hat{x})\theta_j(\hat{x})\mathrm{d}\hat{x}, \tag{4.29}$$

para obtener un número de condición óptimo deberíamos considerar bases jerárquicas de Legendre, que son ortogonales (ortonormales, si las multiplicamos por constantes adecuadas para normalizarlas) y así dan lugar a que $\hat{m}_{ij} = \delta_{ij}$. Aunque es necesario advertir que no es sencillo aplicar este tipo de bases a elementos finitos continuos (y suelen ser utilizadas para elementos finitos discontinuos).

Un problema interesante se plantea al abordar la resolución numérica de EDP que combinan ambos tipos de matrices, de rigidez y de masa (por ejemplo, en la resolución de la ecuación del calor, $u_t - u_{xx} = 0$). Existe de esta forma la posibilidad de, por ejemplo, utilizar técnicas que descompongan la ecuación en términos diferentes[1] (uno de ellos con matriz de masa y otro con matriz de rigidez) para poder aplicar a cada uno de ellos las bases jerárquicas adecuadas.

4.4 Diseño de funciones base

Los conjuntos de funciones de forma de orden alto tanto nodales como jerárquicas que acabamos de ver en la sección anterior, consisten en funciones de forma de tipo sombrero o de tipo burbuja. Las funciones sombrero consisten en funciones que son distinto de cero

[1] Técnicas de *splitting* o descomposición de operadores

en algunos de los puntos extremo del intervalo de referencia $\widehat{K} = [-1, 1]$ y 0 en el otro, mientras que las funciones burbuja son aquellas que se anulan en ambos extremos del intervalo.

4.4.1 Funciones sombrero

Las funciones sombrero están relacionadas con los puntos del mallado y su soporte está formado por dos elementos seguidos. Consideremos el punto x_{i+1} y los elementos adyacentes K_i y K_{i+1}. En el caso jerárquico la función base correspondiente, v_i, viene definida como

$$v_i(x) = \begin{cases} (l_1 \circ F_{K_i}^{-1})(x), & x \in K_i, \\ (l_0 \circ F_{K_{i+1}}^{-1})(x), & x \in K_{i+1}. \end{cases} \tag{4.30}$$

En el caso de elementos nodales, la función base correspondiente v_i viene dada por

$$v_i(x) = \begin{cases} (\theta_{p+1} \circ F_{K_i}^{-1})(x), & x \in K_i, \\ (\theta_1 \circ F_{K_{i+1}}^{-1})(x), & x \in K_{i+1}, \end{cases} \tag{4.31}$$

donde p es el orden del polinomio asociado tanto a K_i como a K_{i+1}.

4.4.2 Funciones burbuja

Las funciones burbuja para un elemento K_i de orden p se define de forma análoga como la composición de una función burbuja y la aplicación inversa $F_{K_i}^{-1}$. En el caso jerárquico usamos las funciones burbuja $l_2, l_3, ..., l_p$ y en el caso nodal usamos las funciones $\theta_2, \theta_3, ..., \theta_p$.

4.5 Estimaciones de error

Una vez visto como aproximar nuestro problema por un problema discreto para así poder hallar la solución, vamos a ver el error que se considera entre ambas soluciones. La idea es que este error sea lo más pequeño posible, ya que queremos que la solución aproximada sea lo más parecida a la solución real.

Para ello, empecemos viendo la relación de ortogonalidad existente entre la solución exacta u y la aproximada $u_{h,p}$. Sea $V = H_0^1(\Omega)$ y $V_{h,p}$ definida en (4.11).

En primer lugar, tenemos que $u \in V$ es tal que

$$a(u, v) = (f, v), \text{ para todo } v \in V, \tag{4.32}$$

donde $a(u,v) = \int_a^b u'(x)v'(x)\mathrm{d}x$ y $(f,v) = \int_a^b f(x)v(x)\mathrm{d}x$. Por otro lado, $u_{h,p} \in V_{h,p} \subset V$ es tal que

$$a(u_{h,p}, v_{h,p}) = (f, v_{h,p}), \text{ para todo } v_{h,p} \in V_{h,p}. \tag{4.33}$$

Por tanto, como $a(\cdot,\cdot)$ es una forma bilineal se verifica que

$$a(u - u_{h,p}, v_{h,p}) = 0, \text{ para todo } v_{h,p} \in V_{h,p}. \tag{4.34}$$

Ahora bien, si tomamos la norma en H^1 (ver capítulo (1)) obtenemos

$$\begin{aligned}
||u - u_{h,p}||_{H^1}^2 &= a(u - u_{h,p}, u - u_{h,p}) = \\
&= a(u - u_{h,p}, u - v_{h,p} + v_{h,p} - u_{h,p}) = \\
&= a(u - u_{h,p}, u - v_{h,p}) + a(u - u_{h,p}, v_{h,p} - u_{h,p}) = \quad (\text{ya que } v_{h,p} - u_{h,p} \in V_{h,p}) \\
&= a(u - u_{h,p}, u - v_{h,p}) \leq \quad (\text{por la desigualdad de Cauchy-Schwarz (1.3)}) \\
&\leq ||u - u_{h,p}||_{H^1} ||u - v_{h,p}||_{H^1}.
\end{aligned}$$

En el caso en que $||u - u_{h,p}||_{H^1} \neq 0$ se tiene que $||u - u_{h,p}||_{H^1} \leq ||u - v_{h,p}||_{H^1}$.

Sin embargo, si $||u - u_{h,p}||_{H^1} = 0$ estamos ante un caso trivial ya que no habría error. Por ello, podemos considerar que $||u - u_{h,p}||_{H^1} \leq ||u - v_{h,p}||_{H^1}$. Tomando el ínfimo se tiene

$$||u - u_{h,p}||_{H^1} \leq \inf_{v_{h,p} \in V_{h,p}} \{||u - v_{h,p}||_{H^1}\}, \tag{4.35}$$

y como $u_{h,p} \in V_{h,p}$

$$||u - u_{h,p}||_{H^1} \geq \inf_{v_{h,p} \in V_{h,p}} \{||u - v_{h,p}||_{H^1}\}. \tag{4.36}$$

Entonces,

$$||u - u_{h,p}||_{H^1} = \inf_{v_{h,p} \in V_{h,p}} \{||u - v_{h,p}||_{H^1}\}. \tag{4.37}$$

Además, como existe un elemento en el que se alcanza el ínfimo ($u_{h,p}$), podemos hablar de mínimo:

$$||u - u_{h,p}||_{H^1} = \min_{v_{h,p} \in V_{h,p}} \{||u - v_{h,p}||_{H^1}\}. \tag{4.38}$$

Observación 4.10. La ecuación (4.38) nos da una estimación de error para el método de Galerkin. Además, podemos concluir que la norma H^1 nos da el error óptimo.

Veamos ahora el error en norma L^2:

Sea w la solución del problema

$$-w'' = u - u_{h,p} \text{ en } [a,b] \text{ con } w(a) = w'(b) = 0. \tag{4.39}$$

Integrando por partes y denotando $(\cdot,\cdot)$ al producto escalar en $L^2(\Omega)$,

$$\begin{aligned} ||u-u_{h,p}||^2_{L^2} &= (u-u_{h,p}, u-u_{h,p}) = \\ &= (u-u_{h,p}, -w'') = \\ &= a(u-u_{h,p}, w) = \\ &= a(u-u_{h,p}, w) - a(u-u_{h,p}, v_{h,p}) = \\ &= a(u-u_{h,p}, w-v_{h,p}), \quad \forall v_{h,p} \in V_{h,p}. \end{aligned}$$

Por la desigualdad de Cauchy-Schwarz (1.3) obtenemos

$$||u-u_{h,p}||_{L^2} \leq \frac{||u-u_{h,p}||_{H^1}||w-v_{h,p}||_{H^1}}{||u-u_{h,p}||_{L^2}} = \frac{||u-u_{h,p}||_{H^1}||w-v_{h,p}||_{H^1}}{||w''||_{L^2}}. \tag{4.40}$$

Tomando de nuevo el ínfimo en $v \in V_{h,p}$ tenemos

$$||u-u_{h,p}||_{L^2} \leq ||u-u_{h,p}||_{H^1} \inf_{v_{h,p}\in V_{h,p}} \frac{||w-v_{h,p}||_{H^1}}{||w''||_{L^2}}. \tag{4.41}$$

Asumimos que w puede ser aproximada por una función de $V_{h,p}$. Por ello, tomamos $v_{h,p} \in V_{h,p}$ cercano a w, teniendo así la suposición de aproximación.

$$\inf_{v_{h,p}\in V_{h,p}} ||w-v_{h,p}||_{H^1} \leq h||w''||_{L^2}, \tag{4.42}$$

donde $h > 0$ es lo más pequeño posible. Esta suposición de aproximación se verifica en particular para elementos finitos de orden 1 ($p = 1$). Entonces

$$||u-u_{h,p}||_{L^2} \leq h||u-u_{h,p}||_{H^1}. \tag{4.43}$$

Si suponemos que u también verifica la suposición de aproximación tenemos

$$||u-u_{h,p}||_{H^1} \leq h||u''||_{L^2}. \tag{4.44}$$

De esta forma, obtenemos el siguiente resultado:

Teorema 4.11. Sea $u \in V$ que verifica (4.32) y $u_{h,p} \in V_{h,p}$ con $p = 1$ que verifica (4.33) entonces

$$||u - u_{h,p}||_{L^2} \leq C_1 h^2, \tag{4.45}$$

$$||u - u_{h,p}||_{H^1} \leq C_2 h, \tag{4.46}$$

con C_1, C_2 constantes positivas.

Observación 4.12. Este teorema significa que para elementos con $p = 1$ podemos esperar errores de orden 2 para la norma de L^2 y de orden 1 para la norma de H^1.

Además, lo anterior se puede generalizar en el siguiente sentido: Para elementos finitos de orden p se puede esperar orden $p+1$ para la norma de L^2 y orden p para la norma de H^1. Esta afirmación depende de la regularidad de los datos del problema. Para una exposición rigurosa de la misma, se puede ver por ejemplo [5] o [3].

CAPITULO

5

Experimentos numéricos

Una vez estudiado el planteamiento teórico de este trabajo nos disponemos a hacer diversos ejemplos en los que comprobamos que se verifica lo estudiado en la teoría. Para ello, hemos programado una biblioteca en Fortran/Python llamada *libHOFFE* en la que se pone en practica el uso de los polinomios tanto de Lagrange como de Lobatto.

5.1 La biblioteca *libHOFFE*

La biblioteca *libHOFFE*, que como su nombre indica es una biblioteca en Fortran de elementos finitos de orden alto (Higher Order Fortran Finite Element library), nace de la idea de aprender los algoritmos que la componen, así como los lenguajes de programación Fortran y Python.

El núcleo de la biblioteca está programada en Fortran ya que, a diferencia de Mathematica o Python, es un lenguaje de programación compilado lo cual permite crear programas de mayor eficiencia computacional. A esto se une, el hecho de que históricamente Fortran se diseñó específicamente pensando en el desarrollo de aplicaciones de alto rendimiento y desde sus orígenes ha sido usado para la resolución numérica de sistemas de ecuaciones de grandes dimensiones. En sus últimas versiones (en la actualidad, la más reciente es Fortran2008) el lenguaje ha incorporado muchas de las características de los lenguajes de programación modernos (entre ellas, la manipulación directa de *arrays* y la orientación a objetos) y, en la actualidad Fortran es, junto a C/C++, el lenguaje de referencia para la

programación científica de altas prestaciones. Aparte de esto, Fortran permite de forma cómoda la programación en paralelo (utilizando nativamente la mayor parte de los esquemas de cálculo paralelo de la actualidad) e incluso, como podemos ver, con un mínimo esfuerzo podemos comenzar a explotar esta técnica para obtener mejores resultados.

La técnica de paralelización consiste en distribuir el programa en diferentes partes, haciendo que cada una de ellas sea resuelta por un núcleo del procesador (o incluso por procesadores localizados en distintas máquinas). La idea es, por tanto, trabajar con varias secuencias a la vez, mejorando así el tiempo computacional empleado. Esta técnica no es sencilla, puesto que programar en paralelo de forma óptima requiere de un gran esfuerzo. Hay que tener en cuenta que el hecho de separar secuencias en distintos núcleos para posteriormente volver a juntarlas, implica un coste computacional no despreciable. Por ello, es imprescindible que la ganancia con la técnica del paralelismo sea suficiente para compensar esta pérdida. Dentro de los métodos más usados para paralelización en Fortran a nivel de CPU, distinguimos OpenMP y MPI. Vamos a hacer una breve introducción a éstos aunque, en cualquier caso, para más información consultar, por ejemplo, [11].

OpenMP es un método de paralelización, usado para ordenadores usuales, con memoria compartida. Considerando la problemática que cualquier método de paralelización conlleva, OpenMP es bastante sencillo de usar puesto que se trata de poner en forma de comentarios, una serie de instrucciones como podremos ver más adelante.

MPI es un método más complejo que OpenMP. Se aplica a supercomputadores, con memoria distribuida y muchos núcleos. De esta forma en un mismo instante se pueden estar realizando muchísimos procesos a la vez. Aunque el hecho de trabajar con memoria distribuida complica mucho la implementación del método.

Además del núcleo en Fortran que hemos comentado anteriormente, se ha desarrollado una interfaz en lenguaje Python que permite acceder a las funciones del núcleo Fortran con mayor agilidad, para así poder desarrollar ejemplos, hacer gráficas, etc... Esta conexión es posible gracias a f2py, un programa que automatiza la generación de interfaces python/-numpy a bibliotecas Fortran. En concreto, el programa trata de extender Python para crear un módulo que acceda a la biblioteca Fortran. De aquí nace el módulo Python que hemos desarrollado, y al que hemos denominado *pyhoffe*. Éste sirve de enlace con Fortran para permitir el uso de los ejemplos en Python.

Además de todo esto, desde el principio hemos estado usando un sistema de control de versiones conocido como *Git*. Este tipo de sistemas permite el trabajo en grupo de for-

ma automática ya que todo el código fuente se guarda en un repositorio remoto que se puede descargar desde un ordenador u otro. El sistema de control de versiones se encarga, entre otras cosas, de almacenar una historia de cada fichero (permitiendo la vuelta a versiones anteriores) y de integrar las versiones de los diferentes usuarios. Para el desarrollo del software, incluyendo el trabajo con este sistema de control de versiones hemos usado la Shell de Unix mediante la cual usando git hemos podido trabajar de forma rápida y cómoda. Aprender a usar esto no es inmediato, pero con un poco de esfuerzo se pueden obtener resultados muy satisfactorios. El compilador Fortran usado ha sido *gfortran*. Como herramienta de compilación se ha utilizado *Make*.

El código de esta biblioteca es de uso libre y se puede consultar en el siguiente repositorio de *Bitbucket* (un sitio web que permite alojar proyectos git):

https://bitbucket.org/EstefaniaAlonso1234/libhoffe.git.

5.2 Problema de Poisson

En primer lugar, consideramos un ejemplo sencillo para poder comprobar que nuestra biblioteca nos da los resultados esperados. De esta forma procedemos a resolver la ecuación

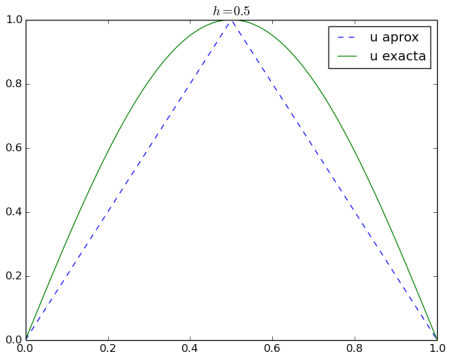
$$u'' = u_0, \tag{5.1}$$

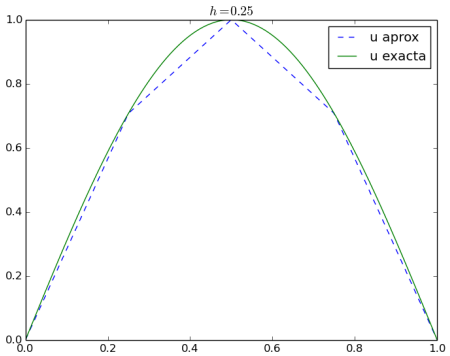
en el dominio $\Omega = (0, 1)$ junto con las condiciones de contorno de tipo Dirichlet homogéneas

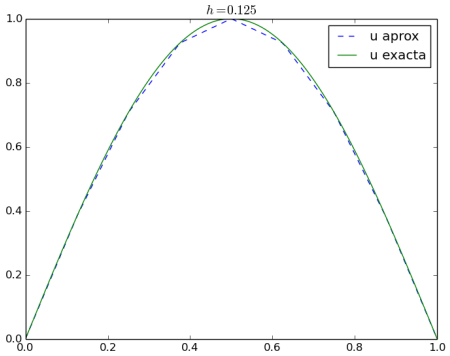
$$u(a) = u(b) = 0. \tag{5.2}$$

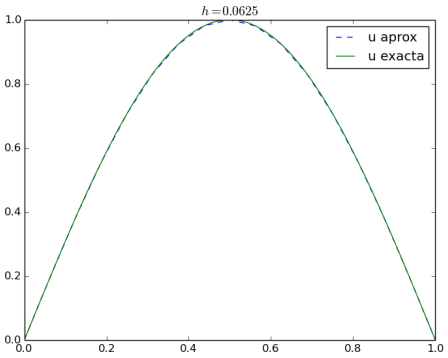
En este caso, consideramos la solución exacta $u = sen(\pi x)$ para poder compararla con la solución aproximada que obtenemos. Para el cálculo de la solución aproximada, hacemos una partición del dominio en distintos nodos equiespaciados, con espacio h. Hemos hecho el estudio para distintos valores de h de forma que vemos que cuanto más fina es la discretización, mejor se ajusta nuestra solución aproximada a la solución exacta (el código se puede consultar en A.1).

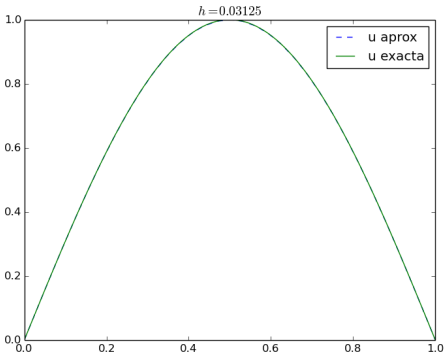
Analizamos lo anterior en las siguientes gráficas donde vamos pintando tanto la solución exacta como la aproximada para los valores $h = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$.

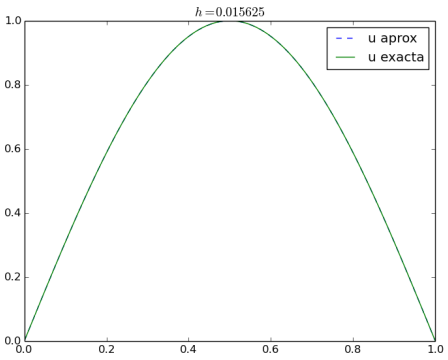

$h = 0.5$
u aprox
u exacta
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


$h = 0.25$
u aprox
u exacta
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

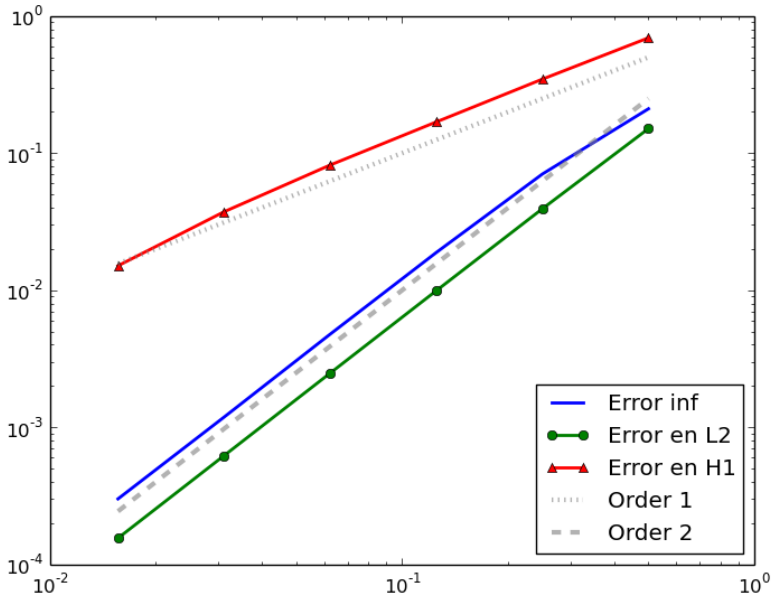

$h = 0.125$
u aprox
u exacta
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0







Una vez vista la comparación entre ambas soluciones, podemos hacer el análisis del error. Para ello, calculamos los errores en norma infinito, en norma L2 y en norma H1, obteniendo de esta forma la siguiente gráfica loglog:



Aún así, para un análisis más claro vamos a ver una tabla con los errores:

Norma	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
∞	1,581	1,901	1,979	2,009	1,982
L^2	1,942	1,985	1,996	1,999	2,000
H^1	1,001	1,036	1,044	1,132	1,308

Efectivamente, obtenemos los resultados que ya habíamos visto de forma teórica en la sección (4.5), ya que en norma L^2 tenemos orden 2 mientras que en norma H^1 tenemos orden 1.

5.3 Ecuación del calor

Una vez hemos visto que nuestra biblioteca funciona bien y que los resultados que salen son los esperados podemos programar ejemplos más avanzados. Para ello, resolvemos la ecuación del calor definida, para $x \in \Omega = (0, 1)$ y para el intervalo temporal $[0, 1]$, por

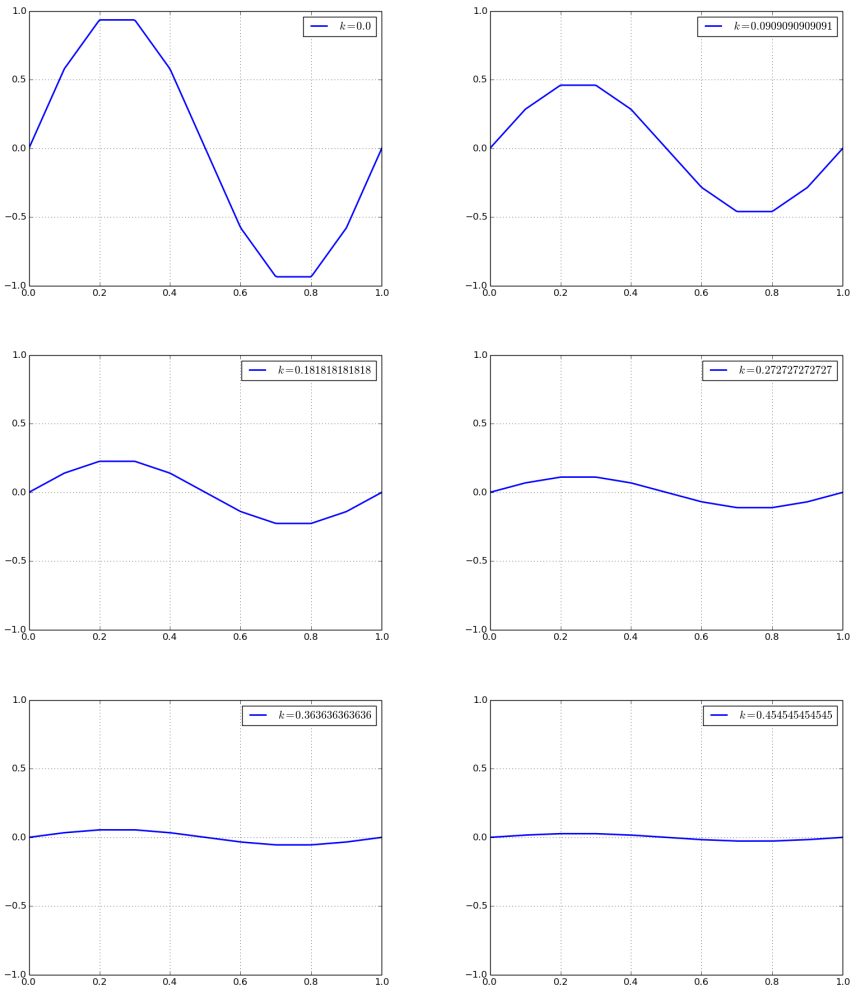
$$u_t - \nu u'' = 0, \tag{5.3}$$

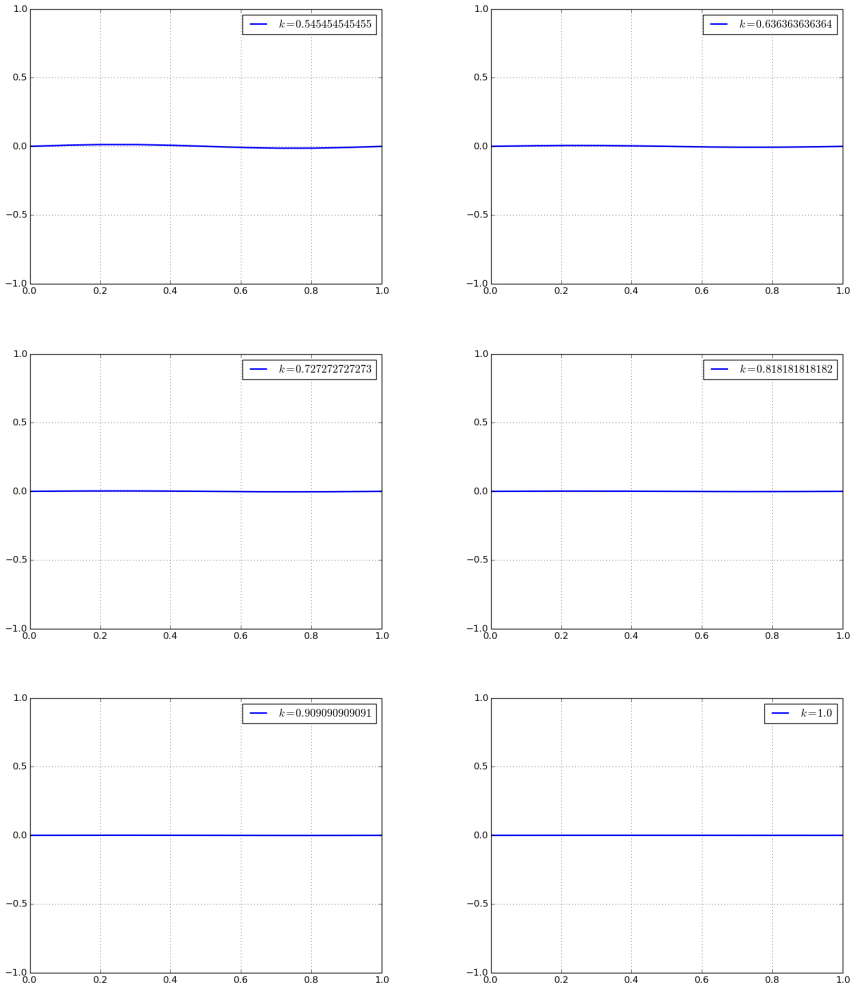
junto a una condición inicial $u|_{t=0} = u^0(x)$ (que, en este caso, viene dada por $u^0(x) = \text{sen}(2\pi x)$) y condiciones de contorno de tipo Dirichlet homogéneas (5.2).

Para resolver este problema, que es de tipo evolutivo, hacemos diferencias finitas en tiempo. En concreto, dividimos el intervalo temporal en subintervalos de tamaño $k \in \mathbb{R}$ y aproximamos u_t mediante diferencias finitas regresivas: $u_t(t_n) \approx \frac{u_n - u_{n-1}}{k}$. Sustituyendo en la ecuación obtenemos el esquema iterativo: Dado $u_0 = u^0$, calcular u_n como solución de:

$$u_n - \alpha u_n'' = u_{n-1},$$

junto a (5.2), con $\alpha = k\nu$. Este problema ya es de tipo estacionario (independiente del tiempo) y lo resolvemos con elementos finitos de Lagrange de orden $p = 1$. El código en Python lo podemos ver en A.2. Partiendo de la solución exacta $u = sen(2\pi x)$ obtenemos las siguientes soluciones del problema para cada instante de tiempo.





El resultado coincide con lo esperado en la práctica, pues se observa la difusión con el tiempo de la situación inicial. Por ejemplo, si la situación inicial fuera de una barra a la que, en el momento inicial, le aportamos distintos niveles de calor en cada zona, este calor tiende a ir disminuyendo conforme pasa el tiempo y al final termina todo equilibrandose.

5.4 Ecuación de transporte

Tras estudiar la ecuación del calor, podemos estudiar también la ecuación de transporte, en el mismo dominio espacial y temporal, que viene definida por

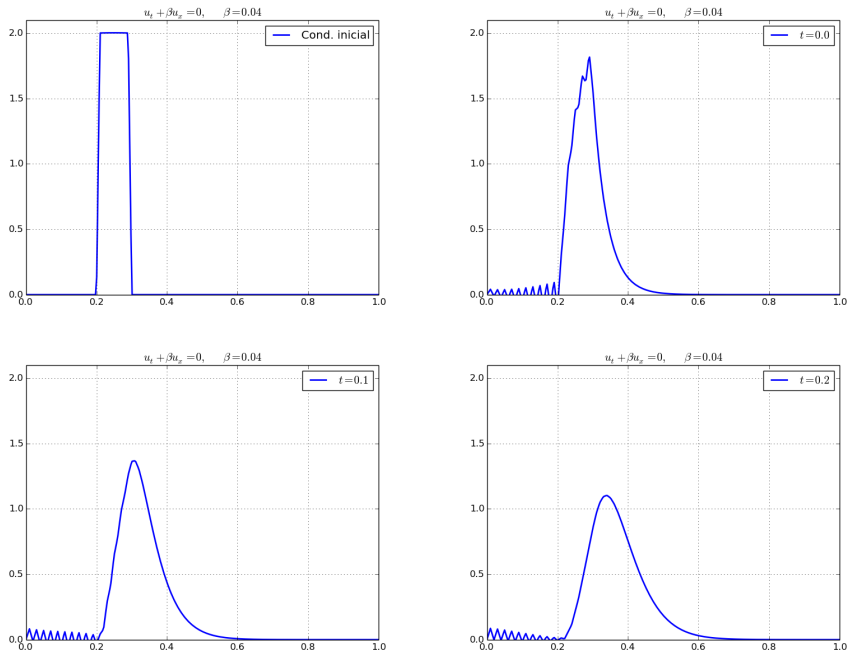
$$u_t - \beta u' = 0 \tag{5.4}$$

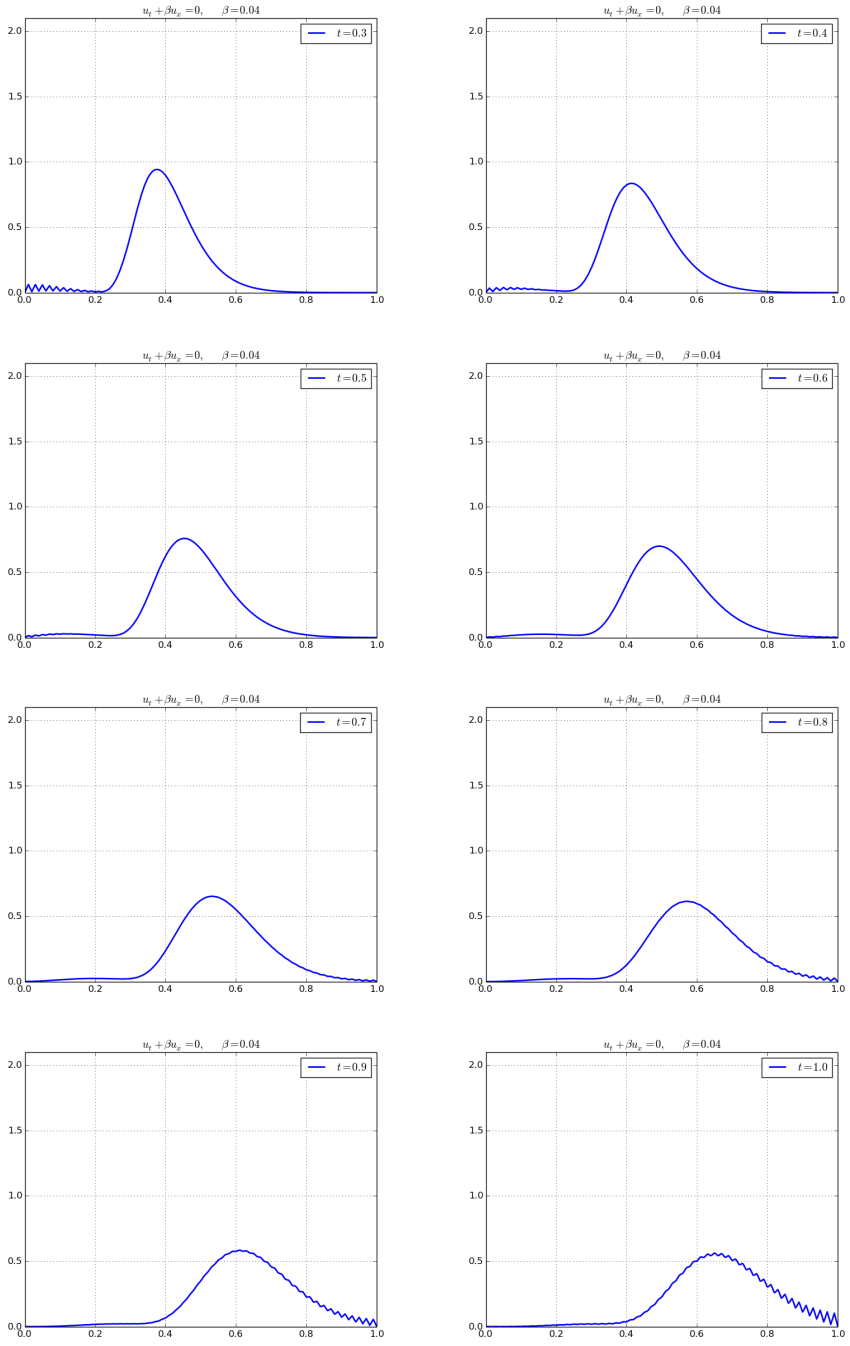
junto a (5.2) y la condición inicial definida posteriormente. La constante $\beta > 0$ puede interpretarse como la velocidad de transporte de la magnitud definida por u.

Al igual que ocurría en el ejemplo anterior, si queremos resolver el problema asociado a esta ecuación junto con las condiciones de contorno de tipo Dirichlet correspondientes, hacemos diferencias finitas regresivas en tiempo. De esta forma obtenemos un problema que ya no depende del tiempo:

$$u_n - \alpha u'_n = u_{n-1}, \tag{5.5}$$

junto a (5.2), con $\alpha = k\beta$. Resolvemos el problema con elementos finitos de Lagrange, partiendo de la condición inicial $f(x) = \text{máx}(1, 2 - (x - x_1)(x - x_2))$ con x_1 y x_2 dos valores definidos previamente en $[0, 1]$ y observando lo que ocurre a medida que pasa el tiempo. Poner esta condición inicial se puede interpretar como la simulación de un atasco que inicialmente está situado entre los puntos x_1 y x_2, para ver como se irá desarrollando a medida que pasa el tiempo. El código de este problema se puede consultar en A.3.





Como podemos ver, en este problema se producen grandes inestabilidades (aparecen picos en las gráficas). Esto es lógico pues el método de los elementos finitos no está, en general, bien adaptado a la resolución de EDP hiperbólicas, como anterior. Y, de hecho,

hacer elementos finitos en dimensión uno (con una partición uniforme) es equivalente a hacer diferencias finitas centradas que, como es bien conocido, dan inestabilidades para este problema. Para más detalles sobre este asunto, ver [12].

Entre las posibles soluciones para evitar la aparición de inestabilidades nos encontramos por un lado el posible uso de elementos finitos discontinuos, para los cuales son unas buenas candidatas las bases de Legendre, pero su resolución se escapa de las pretensiones de este trabajo. Por otro lado, podemos hacer uso del método de difusión artificial, que consiste en introducir un término difusivo a la ecuación pudiendo así resolverla sin que aparezcan inestabilidades (aunque introduciendo un "error de difusión" que podría no tener significado físico).

5.5 Ecuación de convección-difusión

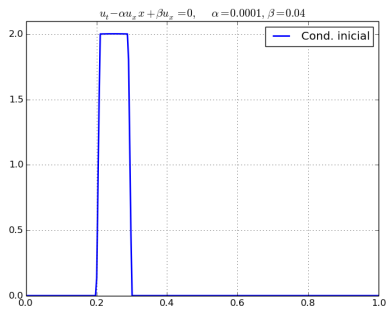
Para solucionar los problemas de la ecuación de transporte, se puede usar el método de difusión artificial (ver [12]). Como hemos dicho, lo que hacemos es añadir un término difusivo, $-\nu u''$ (donde $\nu > 0$ es la constante de difusión o de viscosidad), a la ecuación de convección de la que partíamos. Así, el problema a resolver es (para las condiciones de contorno e inicial anteriores):

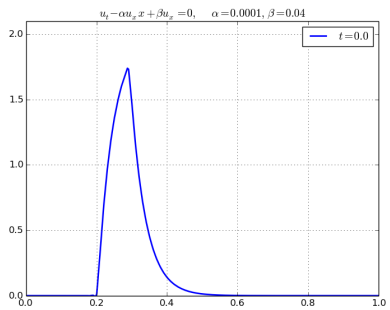
$$u_t - \nu u'' + \beta u' = 0. \tag{5.6}$$

De nuevo, aplicamos diferencias finitas en tiempo para obtener una nueva ecuación estacionaria,

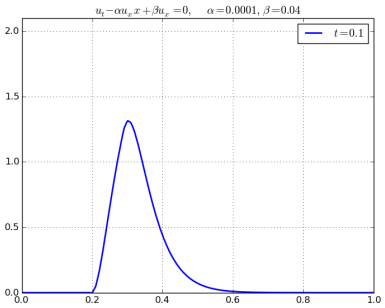
$$u_n - \alpha u_n'' + \hat{\beta} u_n' = u_{n-1}, \tag{5.7}$$

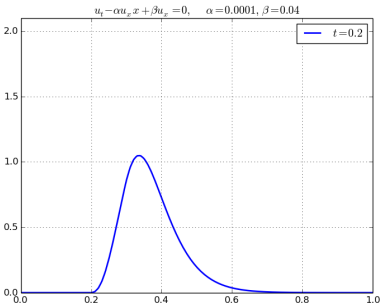
donde $\alpha = k\nu$ y $\hat{\beta} = k\beta$. Resolviendo esta ecuación de forma numérica (ver código en A.4) obtenemos:

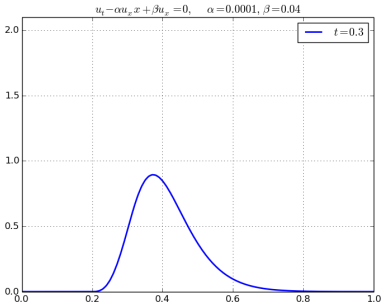


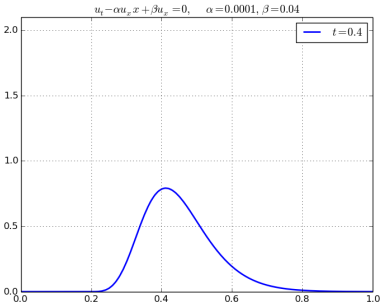


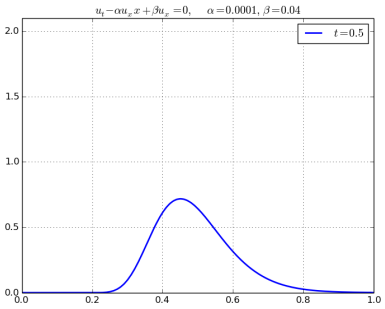
5. EXPERIMENTOS NUMÉRICOS

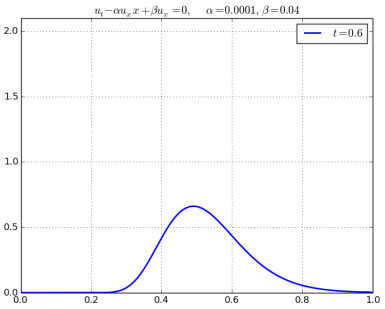


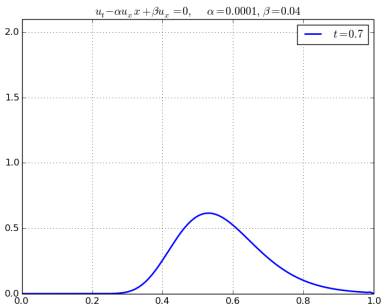


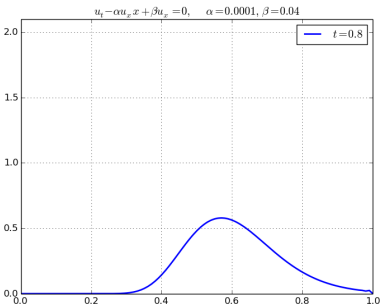


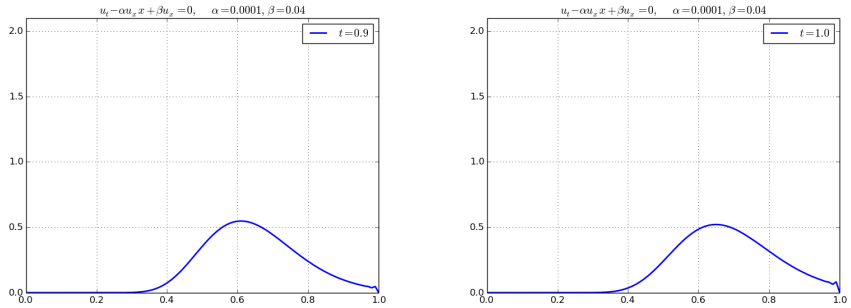










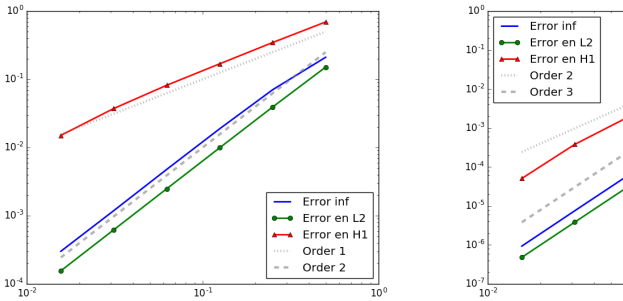


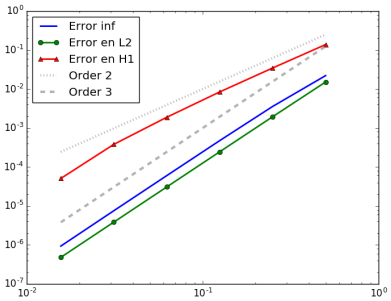
Aquí podemos observar que hemos mejorado las inestabilidades que nos salían. Por ello, la solución es adecuada para resolver el problema que nos planteabamos con la ecuación de transporte.

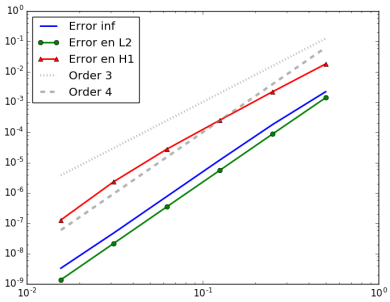
Además de este uso matemático, hay que tener en cuenta que este problema tiene también muchas aplicaciones físicas. Un ejemplo de estas aplicaciones físicas puede ser el estudio del tráfico en una autopista (es el caso que más se adapta al problema concreto que hemos resuelto) aunque también existen numerosas aplicaciones en biología, medicina, economía...

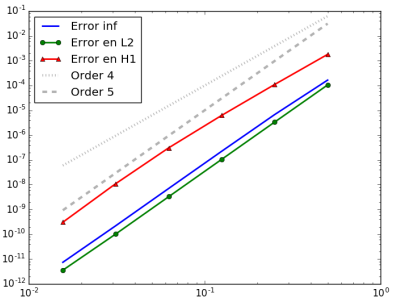
5.6 Problema de Poisson con bases de Lobatto de orden alto

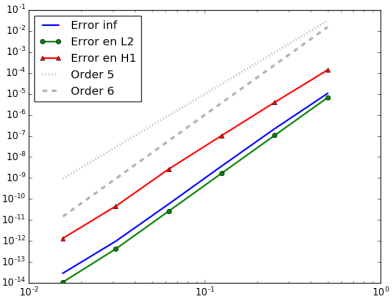
En este ejemplo vamos a estudiar de nuevo la ecuación de Poisson que teníamos en el primer problema (5.1), pero lo vamos a hacer con elementos finitos de orden alto. El código correspondiente lo encontramos en A.5. Veamos de esta forma los órdenes de error correspondiente al orden de los polinomios, en concreto, consideramos $p = 1, ..., 7$.

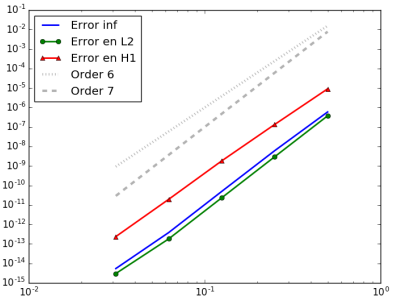


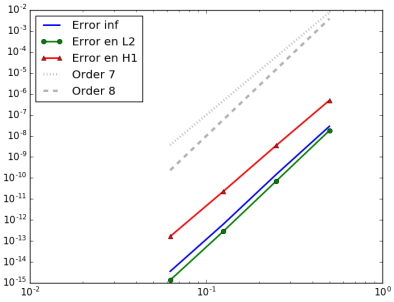












Efectivamente, conforme vamos aumentando el orden de los polinomios va disminuyendo el error. En particular, podemos ver que cuanto más nos acercamos a orden 7, los erres que se alcanzan son menores. Para $p = 5$ estamos cerca de 10^{-15}, es decir, del 0 de la

máquina, con $h \approx 10^{-2}$. Y, para $p = 7$, ¡alcanzamos el cero de la máquina tomando solamente con 16 elementos (esto es, $h \approx 10^{-1}$)! Esto se traduce en que se alcanza la solución exacta para la precisión de la máquina.

Debe observarse que la implementación en el ordenador de este tipo de bases no es tan sencilla como en las funciones de Lagrange con $p = 1$ usadas en los ejemplos anteriores. Por ejemplo, la solución obtenida en el ordenador (que es la solución de un sistema de ecuaciones con una matriz de rigidez global y por tanto viene dada como un vector), no coincide con los valores de la solución sobre los nodos de una partición de $\Omega = (0, 1)$, sino con las "coordenadas" de la solución para la base de Lobatto de cada subintervalo (ver la definición de bases jerárquicas, en el capítulo 3). Por tanto, si queremos calcular el valor de la solución en los puntos es necesario evaluar, en cada subintervalo, la combinación lineal dada por las coordenadas obtenidas y por las funciones base de Lobatto.

Pero, eso sí, una vez se han implementado correctamente este tipo de elementos para orden $p = 2$, la extensión a órdenes superiores, $p = 3, 4, 5, ...$ es muy sencilla: los algoritmos no tienen dificultades adicionales y, para cada p, la base de de Lobatto puede ser reutilizada para orden $p + 1$.

5.7 Cálculo paralelo con bases de Lobatto de orden alto

Por último, resolvemos el mismo problema (5.1) pero esta vez usando técnicas de paralelización. Hay que tener en cuenta que, a diferencia de los anteriores, para usar paralelización el ejemplo está programado en Fortran y no en Python. La idea de usar técnicas de paralelización parte del hecho de que con elementos de Lobatto teníamos una matriz local casi diagonal (ver 4.5). Al hacer el montaje de la matriz global tenemos entonces una matriz de la siguiente forma:

$$S = \begin{bmatrix} s_{11} & s_{12} & 0 & 0 & ... & 0 & 0 & ... & ... & 0 \\ s_{21} & s_{22} & s_{23} & 0 & ... & 0 & 0 & ... & ... & 0 \\ 0 & s_{32} & s_{33} & s_{34} & ... & 0 & 0 & ... & ... & 0 \\ 0 & 0 & s_{43} & s_{44} & ... & 0 & 0 & ... & ... & 0 \\ \vdots & & & & \ddots & & & & & \vdots \\ 0 & 0 & 0 & 0 & ... & s_{jj} & 0 & ... & ... & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & s_{j+1,j+1} & ... & ... & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & ... & \ddots & ... & 0 \\ \vdots & & & & & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & ... & ... & s_{nn} \end{bmatrix},$$

Así tenemos por un lado una matriz tridiagonal de dimensión (ncells+1)×(ncells+1) y por otro lado una matriz diagonal de dimensión ncells(p-1)×ncells(p-1), donde ncells es el número de intervalos de la partición. Lo que hacemos es que cada uno de los sistemas asociados a estas matrices serán resueltas por un núcleo del procesador. En concreto, para resolver el sistema tridiagonal se usará el método de Crout (que consiste en el método de factorización LU para matrices tridiagonales), mientras que la resolución del sistema diagonal es trivial.

Otra de las propiedades de este tipo de matriz es que, como se puede ver de forma clara en la matriz diagonal del sistema, las dos primeras filas aparecen siempre sea cual sea el orden de los polinomios (y su tamaño va a depender sólo del número de celdas), es decir, en la matriz global la matriz tridiagonal va a aparecer siempre pero, si no aumentamos el número de celdas, su tamaño no cambia. Sin embargo, las dimensiones de la matriz diagonal dependerán del orden de los polinomios y así el único coste de aumentar el orden es incrementar la dimensión de una matriz que es diagonal.

Así, el hecho de que la matriz tridiagonal tenga que ser resuelta siempre y que lo único que cambie de un orden de polinomios a otro es la matriz diagonal, hará que el tiempo de computación no incremente demasiado, pues resolver un sistema que está formado por una matriz diagonal, casi no implica coste computacional.

De esta forma, ejecutando el código correspondiente, el cual podemos ver en A.6, tenemos que los tiempos para la resolución del sistema de ecuaciones para $p = 1, ..., 7$ son:

Orden de los polinomios	Tiempo de cálculo [1]
1	0.184000000 s
2	0.195999980 s
3	0.251999974 s
4	0.192000151 s
5	0.196000099 s
6	0.252000332 s
7	0.203999996 s

Comprobamos que, como habíamos previsto, los tiempos de cálculo no incrementan mucho conforme aumentamos el orden de los polinomios.

[1] Tiempo de cálculo en un ordenador con procesador i7 de 4 núcleos y 16Gb de RAM.

Conclusiones y proyectos futuros

En este trabajo hemos estudiado de forma teórica las nociones relacionadas tanto con elementos finitos de Lagrange, para partiendo de ellos analizar los elementos finitos jerárquicos. En el primer capítulo analizamos la teoría variacional de la ecuaciones en derivadas parciales, así como el método de Galerkin, en el que se basa el método de los elementos finitos.

A partir de este planteamiento, hemos podido estudiar las nociones básicas que definen al método. Partiendo de la definición de elementos finitos clásica para EDP elípticas, formulada por Ciarlet [6], llegamos a definiciones menos usuales como puede ser la de conformidad de los espacios o elementos finitos jerárquicos. Hemos estudiado numerosas propiedades de estos elementos, que resumimos a continuación.

Por un lado hemos visto que, para poder aumentar el orden de los polinomios (p–adaptividad) los elementos jerárquicos son buenos candidatos, pues es suficiente con añadir una nueva función a la base que ya teníamos. Sin embargo, para aumentar el orden de los polinomios en los elementos de Lagrange hay que volver a calcular todos los elementos de la base.

Además, los elementos finitos jerárquicos formados por bases de Lobatto, tienen la propiedad particular de que el número de condicionamiento de la matriz de rigidez asociada es mucho mejor que el de la matriz de rigidez asociada a las bases de Lagrange. De hecho, hemos visto que en el caso unidimensional, el número de condición para Lobatto es óptimo. Y además, la matriz de rigidez es "casi diagonal".

Sin embargo, el implementar los elementos finitos jerárquicos no es nada fácil en comparación a los de Lagrange, en los que los grados de libertad nos dan exactamente el valor del polinomio en el nodo correspondiente mientras que en el caso de los elementos jerárquicos nos da los coeficientes de la combinación lineal de los elementos de la base que definen cada polinomio.

Por otro lado, los elementos finitos con bases de Legendre tienen buenas características para el uso de elementos finitos discontinuos. En los elementos finitos discontinuos no se exige continuidad que entre un elemento y otro, por lo que aparecería un salto. Una de las características de las bases de Legendre es que no contienen bases nodales y por tanto no es sencillo usarlas para definir espacios de elementos finitos continuos, pero son buenos candidatos a la hora de plantearnos la implementación de elementos discontinuos.

Aunque hemos dicho que los elementos finitos de Lagrange son mucho más sencillos de implementar, tienen la desventaja de que usar técnicas de paralalelización no es tan inmediato como puede serlo con los elementos jerárquicos. En éstos se pueden obtener matrices que en algunos casos son casi-diagonales, lo que aporta grandes ventajas de cara a la paralelización.

Por último, hay que tener en cuenta que en caso de que no se desee programar una biblioteca, encontrar software informático dedicado a los elementos finitos jerárquicos no es fácil, ya que la mayoría de ellos están enfocados con exclusividad a los elementos finitos clásicos. Esta es una de las razones por las que la biblioteca *libhoffe*. Hemos realizado numerosos experimentos y comparado que ésta funciona correctamente y que los resultados obtenidos son los esperados de forma teórica.

Todo esto, podemos verlo recogido en la siguiente tabla:

	E.F. Clásicos	E.F. jerárquicos
Fácil implementación	Sí	No
Aumentar el orden de los polinomios	No	Sí
Buenas propiedades condicionamiento M. Rigidez	No	Sí (Lobatto)
Discontinuo	No	Sí (Legendre)
Paralelización	No	Sí
Encontrar software específico	Sí	No

Trabajos futuros

Como dijimos en la introducción, este trabajo se planteó desde el principio con una perspectiva muy amplia e intentando abarcar lo máximo posible. Por ello existen muchas extensiones que se han planteado a corto plazo.

En primer lugar, podemos extender tanto los resultados teóricos como la biblioteca al campo bidimensional. Este paso, no debería ser costoso puesto que toda la teoría estudiada se ha enfocado hacia su posible generalización. Como punto de partida, podemos realizar

una extensión al caso bidimensional utilizando elementos finitos cuadrangulares (productos cartesianos de elementos 1d), que podrían estar implementados en un plazo breve. La extensión a mallas triangulares no es tan sencilla, pero también sería abordable en el futuro. Para más detalles, ver [7].

También podemos profundizar en las técnicas de paralelización para su extensión al montaje de los sistemas pues (por motivos de plazos temporales) en el trabajo actual, estas técnicas sólo se aplicaron para su resolución numérica.

Además, como hemos dicho, sería muy interesante el implementar elementos finitos discontinuos con bases de Legendre. La idea sería estudiarlos con mayor detalle, analizando sus ventajas y sus inconvenientes. Entre las primeras se encuentra la posibilidad de resolver EDP hiperbólicas, como la ecuación de transporte planteada en el capítulo 5, que sería el primer experimento numérico que abordaríamos.

Además de lo anterior, podemos hacer extensiones a medio y largo plazo, profundizando en numerosos aspectos e incluso llegando a niveles de investigación.

Por un lado, sería interesante construir una biblioteca puramente en Python y evaluar el rendimiento frente a la que tenemos en Fortran/Python. Existen bastantes artículos que sugieren que programas Python pueden competir con lenguajes compilados [14].

El manejar los elementos finitos a bajo nivel nos abre la puerta a la posibilidad de trabajar con otro tipo de bases, incluyendo la opción de usar bases que no fueran polinómicas (aunque, probablemente, habría que considerar elementos discontinuos).

Sería interesante implementar otras opciones de paralelización, más allá de OpenMP, como pueden ser MPI en ordenadores con memoria distribuida (como el *cluster* de la Universiad de Cádiz) o incluso programar en GPU (tarjetas gráficas) en lugar de en CPU. Esta última es una opción que cada vez gana mayor peso en el campo de la computación científica. Incluso, las últimas versiones de OpenMP incluyen extensiones para trabajar en GPU.

Finalmente, y de cara a la investigación: sería interesante explorar la posibilidad de abordar sistemas de ecuaciones más complejos y con un sentido físico de gran interés, como las ecuaciones de Navier-Stokes o de explorar trabajos recientes que utilicen métodos de orden alto, como el del artículo [13].

APPENDIX

A

Apéndice 1

En este apéndice encontramos el código fuente asociado a los ejemplos desarrollados en el capítulo (5).

A.1 Código Python: Problema de Poisson

El siguiente fichero Python utiliza la biblioteca libHOFFE (a través del paquete Python pyhoffe) para aproximar la solución del problema de Poisson 1D con condiciones de contorno Dirichlet homogéneas y elementos finitos P1 de Lagrange (o, equivalentemente, elementos P1 con bases de Lobatto).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
################################
# Resuelve el problema u''=u0 #
# con u(a)=bc1 y u(b)=bc2      #
################################
import numpy as np
from numpy.linalg import solve
import pyhoffe
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from numpy import log, sqrt

#definición de funciones a usar
```

```
def plot_orders_line(h, orders=[1,2]):
    # Dibujar una línea recta representando los órdenes de error indicados
    n=len(h)
    x=np.array([h[n-1], h[0]])
    styles=[":","--"]
    i=0
    for o in orders:
        print i
        y=x**o # f(x)=x^o será, en escala logarítmica, una recta con
               # pendiente o
        plt.loglog(x,y,styles[i],color='gray',\
                        lw=3,alpha=0.6,label="Order "+str(o))
        i = i+1

def normaL2_cuadrado(lista_x, lista_y):
    #calcula el cuadrado de la norma L2 para una función sobre una partición
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    suma=0
    for i in range(len(lista_y)-1):
        suma=suma+0.5*(lista_x[i+1]-lista_x[i])* \
                    (lista_y[i+1]**2+lista_y[i]**2)
    return suma

def normaL2(lista_x, lista_y):
    #calcula la norma L2 para una función definida
        #sobre un conjunto de puntos
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    return sqrt(normaL2_cuadrado(lista_x, lista_y))

def normaL2prima_cuadrado(lista_x, lista_y):
    #calcula el cuadrado de la norma L2 para la derivada de una función
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    suma=0
    num=0
    den=0
    for i in range(len(lista_y)-1):
```

```
        num=(lista_y[i+1]-lista_y[i])
        den=(lista_x[i+1]-lista_x[i])
        suma=suma+0.5*(lista_x[i+1]-lista_x[i])*(num/den)**2
    return suma

def normaH1(lista_x,lista_y):
    #calcula la norma H1 para una función definida
        #sobre un conjunto de puntos
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    return sqrt(normaL2_cuadrado(lista_x,lista_y)+ \
                    normaL2prima_cuadrado(lista_x,lista_y))

def uexacta(x):
    #aquí definimos la solución exacta del problema
    #ha de coincidir con la especificada en el fichero "example1.f90"
    return np.sin( np.pi * x)

interval = [0,1] #intervalo sobre el que hacemos la partición
#creamos la partición
listacells = [2,4,8,16,32,64]
listah = [1.0/n for n in listacells]
#definimos listas
vectorerror = []
vectorerrorL2 = []
vectorerrorH1 = []

order=1 #orden de los polinomios
#para cada celda:
for i in range(len(listah)):
    ncell=listacells[i]
    ndofs = ncell+1

    #llamamos a la biblioteca
    pyhoffe.example1_init_lobatto(interval, ncell, order)
    A, b = pyhoffe.example1_build_system(ncell, order)
    #bloqueo de grados de libertad
    tgv = 1e+30
    bc1, bc2 = 0.0, 0.0 # Condiciones de contorno

    A[0,0] = tgv
    A[-1,-1] = tgv
```

```
    b[0] = bc1*tgv
    b[-1] = bc2*tgv
    #resolvemos el sistema de ecuaciones
    u = solve(A,b)

    #partición
    x = np.linspace(0.0, 1.0, ndofs)
    xx = np.linspace(0.0,1.0,200)
    #interpolamos los puntos para hallar la solución aproximada
    uinterp=interp1d(x,u)

    plt.plot(xx, uinterp(xx), "--" ,label="u aprox")
    plt.plot(xx, uexacta(xx), label="u exacta")
    plt.legend()
    plt.title(r"$h=%g$" %(listah[i]))
    plt.show()

    #errores
    error=abs(uexacta(xx)-uinterp(xx))
    errorinf=max(error)
    errorL2=normaL2(xx,error)
    errorH1=normaH1(xx,error)
    vectorerror.append(errorinf)
    vectorerrorL2.append(errorL2)
    vectorerrorH1.append(errorH1)
    print "vectorerror=",vectorerror
    print "vectorerrorL2=",vectorerrorL2
    print "vectorerrorH1=",vectorerrorH1


#grafica loglog
plt.loglog(listah, vectorerror, lw=2, label="Error inf")
plt.loglog(listah, vectorerrorL2, "-o", lw=2, label="Error en L2")
plt.loglog(listah, vectorerrorH1, "-^", lw=2, label="Error en H1")

plot_orders_line( listah, orders=[order,order+1] )

plt.legend(loc="best")
plt.show()

for i in range(len(listah)-1):
    print "i=", i
    p=log(vectorerror[i]/vectorerror[i+1])/log(2)
```

```
print "Orden del método en norma inf es: p=",p
p=log(vectorerrorL2[i]/vectorerrorL2[i+1])/log(2)
print "Orden del método en norma L2 es: p=",p
p=log(vectorerrorH1[i]/vectorerrorH1[i+1])/log(2)
print "Orden del método en norma H1 es: p=",p
```

A.2 Código Python: Ecuación del calor

El siguiente fichero Python emplea la biblioteca libHOFFE (a través del paquete Python pyhoffe) para aproximar la solución de la ecuación del calor 1D con condiciones de contorno Dirichlet homogéneas. Para la discretización tiempo se utilizan diferencias finitas regresivas, mientras que para la discretización en espacio se utilizan se elementos finitos P1 Lagrange (o, equivalentemente, elementos P1 con bases de Lobatto).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
##########################################
# Resuelve el problema                   #
#    u_t - alpha * u'' = 0 en [x_0, x_1]  #
#    con u(x_0)=bc1 y u(x_1)=bc2          #
##########################################
import numpy as np
from numpy.linalg import solve
import pyhoffe
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

#Discretización en tiempo
t_interval=[0.,1.]
t_ncells=10
t_ndof=t_ncells+1
pi=np.pi
#Discretización en espacio
interval = [0.,1.]
ncells  = 10
ndof = ncells+1
x = np.linspace(0.0, 1.0, ndof)
# Definición del problema
def u_exact(x): return np.sin(2*pi*x) # Solución exacta
def d2_u_exact(x): return -(4*pi**2)*np.sin(2*pi*x) # Segunda derivada

uExact = u_exact(np.linspace(interval[0], interval[1], ndof))
```

```
d2uExact = d2_u_exact(np.linspace(interval[0], interval[1], ndof))
print "uExact=", uExact
alpha = 1.0/(4*np.pi**2)
u0 = uExact - alpha*d2uExact
print "u0=", u0
bc1, bc2 = u_exact(interval[0]), u_exact(interval[1])

#lo hacemos en cada instante de tiempo
for n in range(t_ndof+1):

    # Resolución del problema
    A, b = pyhoffe.example2_build_system(interval, u0, alpha, ncells)

    #bloqueo de grados de libertad
    tgv = 1e+30
    A[0,0] = tgv
    A[-1,-1] = tgv
    b[0] = bc1*tgv
    b[-1] = bc2*tgv

    u = solve(A,b)
    xx = np.linspace(0.0, 1.0, 200)
    #interpolación sol.aprox.
    uinterp=interp1d(x,u)
    # Dibujar
    plt.axis([0.0,1.0,-1.0,1.0])
    plt.plot(xx, uinterp(xx), lw=2, label="$k="+str(n*1.0/ndof)+"$")
    plt.legend(loc="best")
    plt.grid()
    plt.show()
    u0=u
```

A.3 Código Python: Ecuación de transporte

El siguiente fichero Python emplea la biblioteca libHOFFE (a través del paquete Python pyhoffe) para aproximar la solución del problema de transporte 1D con condiciones de contorno Dirichlet homogéneas. Para la discretización tiempo se utilizan diferencias finitas regresivas, mientras que para la discretización en espacio se utilizan se elementos finitos P1 Lagrange (o, equivalentemente, elementos P1 con bases de Lobatto).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
#
# ------------------------------------------------------------------------
# Resuelve el problema
#   u_t - alpha * u'' + beta * u' = 0 en [x_0, x_1]
#   + c.c. Dirichlet en x_0 y x_1
# ------------------------------------------------------------------------
import numpy as np
from numpy.linalg import solve
import pyhoffe
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

#Discretización en tiempo
t_interval=[0.,1.]
t_ncells=10
t_ndof=t_ncells+1

# Discretización
interval = [0.,1.]
ncells  = 100
ndof = ncells+1

# Condición inicial
x1=0.2
x2=0.3
def cond_inicial(x):
    # Devuelve 2-(x-x1)*(x-x2) si $x\in (x1,x2)$, cero en otro caso
    return ( 2-(x-x1)*(x-x2) ) * (x1<x)*(x<x2)

def dibujar_solucion(x, u, label):
    #definimos una función para dibujar la solución
    xx = np.linspace(0.0, 1.0, 300)
    #sol.aprox.
    uinterp=interp1d(x,u)
    plt.axis([0.0,1.0,-1.09,1.09])
    plt.plot(xx, uinterp(xx), lw=2, label=label)
    plt.ylim([0,2.1]) # Ajustar el eje y al intervalo [0, 2.1]
    plt.legend(loc="best")
    plt.grid()
    plt.title(r"$u_t+\beta u_x = 0, \quad \beta= %g$" %( beta))
    plt.show()

# Definición del problema
```

```
#conciones de contorno
aplicar_cond_contorno_2 = True # ¿Aplicar la segunda condición de contorno?
bc1 = cond_inicial(interval[0])
if aplicar_cond_contorno_2:
    bc2 = cond_inicial(interval[1]) # Boundary conditions

#coeficientes alfa, beta
k = (t_interval[-1] - t_interval[0]) / t_ncells # Incremento de tiempo
if aplicar_cond_contorno_2:
    alpha = 0
    #si hay c. contorno a izquierda, la velocidad ha de ser pequeña
    beta = 4.e-1 * k
else:
    alpha = 0
    #en otro caso, nos podemos permitir una velocidad mayor
    beta = 1.0e+0 * k

# condición inicial
x = np.linspace(interval[0], interval[1], ndof)
u0 = cond_inicial(x)
dibujar_solucion(x, u0, label="Cond. inicial")

# Resolución del problema mediante iteraciones en tiempo
for n in range(t_ndof):

    A, b = pyhoffe.example6_build_system(interval, u0, alpha, beta, ncells)
    # A no es simétrica. Fortran almacena las matrices por columnas,
    # mientras que Python (y C) las almacena por filas
    A = A.transpose()

    #bloqueo de grados de libertad
    tgv = 1e+30
    A[0,0] = tgv
    b[0] = bc1*tgv
    if aplicar_cond_contorno_2:
        A[-1,-1] = tgv
        b[-1] = bc2*tgv

    u = solve(A,b)

    # Dibujar
    t_actual = t_interval[0] + k*n # Instante de tiempo actual
```

```
    x = np.linspace(0.0, 1.0, ndof)
    dibujar_solucion(x, u, label="$t="+str(t_actual)+"$")

    u0=u
```

A.4 Código Python: Ecuación de convección-difusión

El siguiente fichero Python emplea la biblioteca libHOFFE (a través del paquete Python pyhoffe) para aproximar la solución del problema de convección-difusión 1D con condiciones de contorno Dirichlet homogéneas. Para la discretización tiempo se utilizan diferencias finitas regresivas, mientras que para la discretización en espacio se utilizan se elementos finitos P1 Lagrange (o, equivalentemente, elementos P1 con bases de Lobatto).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# ----------------------------------------------------------------------
# Resuelve el problema
#   u_t - alpha * u'' + beta * u' = 0 en [x_0, x_1]
#   + c.c. Dirichlet en x_0 y x_1
# ----------------------------------------------------------------------
import numpy as np
from numpy.linalg import solve
import pyhoffe
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

#Discretización en tiempo
t_interval=[0.,1.]
t_ncells=10
t_ndof=t_ncells+1

# Discretización
interval = [0.,1.]
ncells  = 100
ndof = ncells+1

# Condición inicial
x1=0.2
x2=0.3
def cond_inicial(x):
    # Devuelve 2-(x-x1)*(x-x2) si $x\in (x1,x2)$, cero en otro caso
```

```
    return ( 2-(x-x1)*(x-x2) ) * (x1<x)*(x<x2)

def dibujar_solucion(x, u, label):
    #definimos una función para dibujar la solución
    xx = np.linspace(0.0, 1.0, 300)
    #sol.aprox.
    uinterp=interp1d(x,u)
    plt.axis([0.0,1.0,-1.09,1.09])
    plt.plot(xx, uinterp(xx), lw=2, label=label)
    plt.ylim([0,2.1]) # Ajustar el eje y al intervalo [0, 2.1]
    plt.legend(loc="best")
    plt.grid()
    #plt.title(r"$u_t - \alpha u_xx + \beta  u_x = 0, \quad \alpha=%g,\
        #\beta= %g$" %(alpha, beta))
    plt.show()

# Definición del problema

#conciones de contorno
aplicar_cond_contorno_2 = True  # ¿Aplicar la segunda condición de contorno?
bc1 = cond_inicial(interval[0])
if aplicar_cond_contorno_2:
    bc2 = cond_inicial(interval[1]) # Boundary conditions

#coeficientes alfa, beta
k = (t_interval[-1] - t_interval[0]) / t_ncells # Incremento de tiempo
if aplicar_cond_contorno_2:
    alpha = 1.e-3 * k
    #si hay c. contorno a izquierda, la velocidad ha de ser pequeña
    beta = 4.e-1 * k
else:
    alpha = 0.e-3 * k
    #en otro caso, nos podemos permitir una velocidad mayor
    beta = 1.0e+0 * k

# condición inicial
x = np.linspace(interval[0], interval[1], ndof)
u0 = cond_inicial(x)
dibujar_solucion(x, u0, label="Cond._inicial")

# Resolución del problema mediante iteraciones en tiempo
for n in range(t_ndof):
```

```
A, b = pyhoffe.example6_build_system(interval, u0, alpha, beta, ncells
# A no es simétrica. Fortran almacena las matrices por columnas,
    # mientras que Python (y C) las almacena por filas
A = A.transpose()

#bloqueo de grados de libertad
tgv = 1e+30
A[0,0] = tgv
b[0] = bc1*tgv
if aplicar_cond_contorno_2:
    A[-1,-1] = tgv
    b[-1] = bc2*tgv

u = solve(A,b)

# Dibujar
t_actual = t_interval[0] + k*n # Instante de tiempo actual
x = np.linspace(0.0, 1.0, ndof)
dibujar_solucion(x, u, label="$t="+str(t_actual)+"$")

u0=u
```

A.5 Código Python: Problema de Poisson con bases de Lobatto de orden alto

El siguiente fichero Python utiliza la biblioteca libHOFFE (a través del paquete Python pyhoffe) para aproximar la solución del problema de Poisson 1D con condiciones de contorno Dirichlet homogéneas y elementos finitos con bases de Lobatto de órden arbitrario (en la versión actual, se admiten polinomios de órdenes comprendidos entre 1 y 7).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import numpy as np
from numpy.linalg import solve
import pyhoffe
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d # Interpolador P1
from scipy.interpolate import lagrange # Interpolador de lagrange
from math import log, sqrt


#=====================================================================
```

A. APÉNDICE 1

```
# Parámetros principales (dominio, nº elementos, orden de polinomios)
#=====================================================================
interval = [0,1]
ncell=10
order=3 # Grados de polinomios, hasta 7.
verbosity=1 # Nivel de verborrea (impresión em pantalla)
#=====================================================================

pyhoffe.example1_init_lobatto(interval, ncell, order)
A, b = pyhoffe.example1_build_system(ncell, order)


#bloqueo de grados de libertad

tgv = 1e+30 # Très grande valeur
i_a = 0 # Índice del grado de libertad para el extremo izqdo.
i_b = ncell # Índice del grado de libertad para el extremo dcho.
bc_a, bc_b = 0.0, 0.0 # Boundary conditions
A[i_a, i_a] = tgv
A[i_b, i_b] = tgv
b[i_a] = bc_a*tgv
b[i_b] = bc_b*tgv

np.set_printoptions(precision=2) # Dígitos decimales
if verbosity>1:
    print "A:\n", A
    print "b:\n", b

# Dibujar errores

def plot_orders_line(h, orders=[1,2]):
    # Dibujar una línea recta representando los órdenes de error indicados
    n=len(h)
    x=np.array([h[n-1], h[0]])
    styles=[":","--"]
    i=0
    for o in orders:
        print i
        y=x**o # f(x)=x^o será, en escala logarítmica,
                    # una recta con pendiente o
        plt.loglog(x,y,styles[i],color='gray',\
            lw=3,alpha=0.6,label="Order "+str(o))
        i = i+1
```

```
# Normas y solución exacta
def normaL2_cuadrado(lista_x, lista_y):
        #calcula la norma L2 al cuadrado de una función
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    suma=0
    for i in range(len(lista_y)-1):
        suma=suma+0.5*(lista_x[i+1]-lista_x[i])*\
                        (lista_y[i+1]**2+lista_y[i]**2)
    return suma
def normaL2(lista_x, lista_y):
        #calcula la norma L2 para la derivada de una función
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    return sqrt(normaL2_cuadrado(lista_x, lista_y))
def normaL2prima_cuadrado(lista_x, lista_y):
    #calcula la norma L2 cuadrado para la derivada de una función
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    suma=0
    num=0
    den=0
    for i in range(len(lista_y)-1):
        num=(lista_y[i+1]-lista_y[i])
        den=(lista_x[i+1]-lista_x[i])
        suma=suma+0.5*(lista_x[i+1]-lista_x[i])*(num/den)**2
    return suma

def normaH1(lista_x,lista_y):
        #calcula la norma H1 de una función
    #argumentos:
    #lista_x[i]=coordenada del punto i-ésimo de la partición
    #lista_y[i]=el valor de la función sobre la coordenada i-ésima
    return sqrt(normaL2_cuadrado(lista_x, lista_y) + \
                normaL2prima_cuadrado(lista_x,lista_y))
def uexacta(x):
    #aquí definimos la solución exacta del problema (que ha de
    #coincidir con la especificada en el fichero "example1.f90"
    return np.sin( np.pi * x)
```

```
listacells = [2,4,8,16,32,64]
listah = [1.0/n for n in listacells]
vectorerror = []
vectorerrorL2 = []
vectorerrorH1 = []

for i in range(len(listah)):
    ncell = listacells[i]

    pyhoffe.example1_init_lobatto(interval, ncell, order)
    A, b = pyhoffe.example1_build_system(ncell, order)
    #bloqueo de grados de libertad
    tgv = 1e+30 # Très grande valeur
    i_a = 0 # Índice del grado de libertad para el extremo izqdo.
    i_b = ncell # Índice del grado de libertad para el extremo dcho.
    bc_a, bc_b = 0.0, 0.0 # Boundary conditions
    A[i_a, i_a] = tgv
    A[i_b, i_b] = tgv
    b[i_a] = bc_a*tgv
    b[i_b] = bc_b*tgv

    u = solve(A,b)
    # if verbosity>1:
        # print "============ u***", u

    global_ndofs = len(u) # Números de grados de libertad globales
    I_a, I_b = interval # Extremos del intervalo
    # Cuidado, si el siguiente parámetro es muy grande
        # se puede colgar el ordenador
    N = 10**3 * ncell+1 # Tamaño de la partición
    x = np.linspace(I_a, I_b, N)
    # Evaluar la solución dada por u en cada punto de x:
    y = pyhoffe.example1_eval_on_partition(x, u, N, global_ndofs)


    xx = np.linspace(I_a, I_b, 200)
    #sol.aprox.
    yinterp=interp1d(x,y)


    #errores
    error=abs(uexacta(xx)-yinterp(xx))
```

```
    errorinf=max(error)
    errorL2=normaL2(xx,error)
    errorH1=normaH1(xx,error)
    vectorerror.append(errorinf)
    vectorerrorL2.append(errorL2)
    vectorerrorH1.append(errorH1)


#grafica loglog
plt.loglog(listah, vectorerror, lw=2, label="Error inf")
plt.loglog(listah, vectorerrorL2, "-o", lw=2, label="Error en L2")
plt.loglog(listah, vectorerrorH1, "-^", lw=2, label="Error en H1")

plot_orders_line( listah, orders=[order,order+1] )

plt.legend(loc="best")
plt.savefig('/tmp/example1-order'+str(order)+'.png')

if verbosity>0:
    for i in range(len(listah)-1):
        print "i= %i (h=%f/%f)" %(i, listah[i], listah[i+1])
        p=log(vectorerror[i]/vectorerror[i+1])/log(2)
        print "Orden del método en norma inf es: p=",p
        p=log(vectorerrorL2[i]/vectorerrorL2[i+1])/log(2)
        print "Orden del método en norma L2 es: print()=",p
        p=log(vectorerrorH1[i]/vectorerrorH1[i+1])\
                /log(listah[i]/listah[i+1])
        print "Orden del método en norma H1 es: p=",p
```

A.6 Código Fortran: Cálculo paralelo con bases de Lobatto de orden alto

El siguiente fichero Fortran utiliza la biblioteca libHOFFE para aproximar la solución del problema de Poisson 1D con condiciones de contorno Dirichlet homogéneas y elementos finitos con bases de Lobatto de órden arbitrario (en la versión actual, se admiten polinomios de órdenes omprendidos entre 1 y 7). La paralelización se realiza, a nivel de la resolución del sistema lineal asociado al método de los elementos finitos, mediante la interfaz estándar de software paralelo *OpenMP*.

```
program example1_openmp
```

```
use nrtypes, only: dp, long
use crout_solver, only: solve => solve_tridiagonal_system
use example1_mod, only:&
     init_lobatto => example1_init_lobatto,&
     build_system => example1_build_system
! use omp_lib ! Incorporar funciones para cálculo paralelo con OpenMP

implicit none

! Declaración de variables
real(dp), allocatable, dimension(:,:) :: A ! Matriz de rigidez
real(dp), allocatable, dimension(:) :: b ! jmiembro
real(dp), allocatable, dimension(:) :: u ! Solución del sistema
integer(long) :: i, ndof
real :: elapsed1, elapsed2, elapsed3, start, finish

! Definición de parámetros
real(dp), dimension(2), parameter :: interval=[0,1]
integer(long), parameter :: order=7 ! Grado de los polinomios
integer(long), parameter :: ncell=2000 ! Número de elementos
real(dp), parameter :: tgv = 1e+30 ! Très grande valeur ;-)
real(dp), parameter :: bc1 = 0.0, bc2 =0.0 ! C.c.
logical :: resul

! Iniciamos elmentos finitos y construimos el sistema de ecuaciones

call init_lobatto(interval, ncell, order) ! Iniciar el ejemplo1

ndof = order*ncell+1 ! Número total de grados de libertad
! print '("ndof = ", I0 )', ndof
allocate(A(ndof,ndof)) ! Dimensionar y alojar A en memoria
allocate(b(ndof), u(ndof)) ! Dimensionar y alojar b y u

! call omp_get_wtime(start)
call cpu_time(start)
call build_system(A, b) ! Construir el sistema en A, b
! call omp_get_wtime(finish)
call cpu_time(finish)
elapsed1 = finish-start

! Bloqueamos los grados de libertad:
     ! grados de libertad en los extremos del intervalo:
associate(i_a => 1, i_b => ncell+1)
```

```
    A(i_a,i_a) = tgv
    A(i_b,i_b) = tgv
    b(i_a) = tgv*bc1
    b(i_b) = tgv*bc2
  end associate

  !omp parallel shared(A, b, u) private(i)
  !omp sections
  ! Calcular la solución para los términos de orden 1
  !omp section
  call cpu_time(start)
  resul = solve(A(1:ncell+1,1:ncell+1), b(1:ncell), u(1:ncell))
  ! u = solución del sistema de ecuaciones Ax = b
  call cpu_time(finish)
  ! print *, "Section 1"
  elapsed2 = finish-start
  if( resul .neqv. .true. ) then
     stop "Sorry the system can not be solved using Crout factorization"
  end if
  ! Calcular la solución para los términos de orden superior
  !omp section
  ! print *, "Section 2"
  call cpu_time(start)
  do i=ncell+2, ndof
     u(i) = b(i)/A(i,i)
  end do
  call cpu_time(finish)
  elapsed3 = finish-start
  !omp end parallel section

  print '("============================================================")'
  print '(" Computed solution with polynomial order=",I0,&
          "                 "(",I0," cells ,",I0," dofs)")',&
        order, ncell,ndof
  ! print '(" Matrix mounting time: ",G0)', elapsed1
  ! print '(" Solving time (tridiagonal system) : ",G0)', elapsed2
  ! print '(" Solving time (diagonal system) : ",G0)', elapsed3
  print '(" Total solving time : ",G0," seconds")', elapsed2 + elapsed3
  print '("============================================================")'


end program example1_openmp
```

Bibliografía

[1] Haim Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer New York, New York, NY, 2010. 5, 6, 8, 9, 10, 11, 12, 13, 24

[2] F. Ortegón Gallego. *Simulación numérica con FreeFem++. Introducción al Método de los Elementos Finitos*. Universidad de Cádiz, 2011. Curso i-math de formación del PDI. 2

[3] Susanne C Brenner and L. Ridgway Scott. *The mathematical theory of finite element methods*. Springer, New York, NY, 2008. 10, 49

[4] F. Hecht. New development in freefem++. *Journal of Numerical Mathematics*, 20(3-4), January 2012. 3

[5] A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Springer, 2004. 14, 25, 49

[6] Philippe G. Ciarlet. *The finite element method for elliptic problems*. Number 40 in Classics in applied mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2002. 17, 67

[7] Pavel Solin, Karel Segeth, and Ivo Dolezel. *Higher-order finite element methods*. Chapman & Hall/CRC, Boca Raton, FL, 2004. 18, 20, 23, 24, 33, 37, 44, 69

[8] Daniele Antonio Di Pietro and Alexandre Ern. *Mathematical aspects of discontinuous galerkin methods*. Springer, Berlin; New York, 2012. 21, 24, 26

[9] Ben Q Li. *Discontinuous finite elements in fluid dynamics and heat transfer*. Springer, London, 2006. 26

[10] J.A.I. Río and J.M.R. Cabezas. *Métodos numéricos: teoría, problemas y prácticas con MATLAB*. Ciencia y técnica Pirámide. Pirámide, 2002. 43

[11] Ian Chivers and Jane Sleightholme. *Introduction to Programming with Fortran*. Springer London, London, 2012. 52

[12] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*, volume 23 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. 61

[13] Zhixing Fu, Luis F. Gatica, and Francisco Javier Sayas. *Matlab tools for hdg in three dimensions.* ACM Trans. Math. Softw., 41(3):20:1–20:21, June 2015. 69

[14] Mikael Mortensen and Hans Petter Langtangen. *High performance python for direct numerical simulations of turbulent flows.* CoRR, abs/1602.03638, 2016. 69

[15] Grégoire Allaire. *Numerical analysis and optimization an introduction to mathematical modelling and numerical simulation*. Oxford University Press, Oxford, 2007.